Graph Decomposition Based on Triangle Count for Community Extraction

Graph Decomposition Based on Triangle Count for Community Extraction

Yanting Li

Department of Artificial Intelligence
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology

This dissertation is submitted for the degree of
Doctor of Philosophy

January 2015

Dedication

I would love to dedicate this dissertation to my father Ji Chuan Li and mother Qi Feng Deng for their love to me. I appreciate them for their effort in providing me with the best possible education, and all wonderful things they have done for supporting my live and study in Japan. In addition, on behalf of my love and respect, I would also love to dedicate this dissertation to my wife Cissy Luk for her selfless understanding, and being my pillars of spiritual support. Her inspiration and strong support have been with me through innumerable days and nights.

Acknowledgements

It is a mighty undertaking to write this dissertation. To accomplish this dissertation and research are tough and isolated. It is hard to work out this dissertation without the support of numerous people. Therefore, my greatest appreciation not only goes to my families, but also goes to these people who provide me with support and wonderful time throughout my study in Kyushu Institute of Technology.

I could not have accomplished the dissertation without the advice and support of Prof. Hiroshi Sakamoto. My sincere gratitude is due first and foremost to Prof. Hiroshi Sakamoto for being unfailingly generous with his patience and time to my study and research. I appreciate Prof. Hiroshi Sakamoto for providing me with the opportunity to deepen my study and research. His patience, innovative ideology, enthusiasm, immense knowledge are enabling me to develop a better understanding of graph theory and algorithm. Prof. Hiroshi Sakamoto also gives me tremendous freedom in pursuing my own ideas when ensuring that I have made progress toward fruitful outcome. His guidance and support are crucial in helping me with writing this dissertation. I am heartly grateful for his guidance and support to my research. Furthermore, I also appreciate Prof. Hiroshi Sakamoto for his kindness in providing financial support to my study and research.

Additionally, I am heartily thankful to Prof. Tetsuji Kuboyama in the Gakushuin Univerisity. I appreciate him for his patient guidance throughout my doctoral course. His constructive suggestion and patient guidance provide my study and research with crucial fundament. Moreover, I appreciate Prof. Tetsuji Kuboyama for his patience and time in reviewing my dissertation, and making suggestions for improvement. These useful suggestions play an essential role in improving the quality of my dissertation. His motivation and expertise are also my precious treasure. I would also like to express my grateful appreciation to Prof. Jong Hoon Huh, Prof. Kouichi Hirata, and Prof. Takeshi Shinohara who are the members of my doctoral committee for their available suggestions and useful reviewing in helping me to improve the quality of my dissertation, and providing a good basis for the present work. I appreciate them so much for putting their works aside and spending their precious time in reviewing the dissertation. Their useful comments and feedback play a crucial role in the modification of the dissertation.

I would like to express my special gratitude to my friends Dr. Adriano Albert Muniz, Mr. Albert Zhang, Mr. Avinash Dev, Dr. Dang Khoa Nguyen, Mr. Guo Qing Zhang, Miss Nikki So, Miss Nan Shen, Mr. Van Hung Tran, and Dr. Xin Zeng for their kindness friendship. I appreciate them for sharing their passionate ideas with me. Their encouragement inspires me a lot when I come across any difficulties in my live and study. Their praiseworthy suggestions bring me so much benefit. There are still many friends who I would like to express my greatest gratitude to them, but too many to list all their names in here. Every little thing they bring me is a gift and precious treasure in my life. Their kindness support, their smile and encouragement inspire me with strengthened confidence to go through all of the difficulties in my life. Besides, I would also like to express my special gratitude to my lab mate Mr. Koji Maeda for helping me within better understanding about the art of program.

Finally, I sincerely offer my best regards and blessing to all of the people who support me in any aspects during my study in Japan. Your selfless support inspires me strongly, and makes my life wonderful.

Yanting Li
January 5^{th}, 2015

Abstract

A community in a graph is a group of nodes holding similar characteristics or sharing common properties within the graph. It is also referred to as a cluster or a module of graphs. A graph is considered to have the community structure if its nodes can be clustered into several subsets of nodes within a certain degree or density. The community structure is, therefore, one of the most relevant characteristics of graphs. The community structure has attracted tremendous interest in recent years for its various scientific applications. The community structure represents real world network systems, and models many kinds of real world relations, such as biological networks, web pages and social networks. Furthermore, the community structure can be used for analyzing the hierarchical organizations of real world network systems, and for understanding the structure of complicated network systems. Various efficient and effective techniques have been proposed and developed for identifying communities in graphs.

Among various community extraction and graph analysis techniques, graph decomposition techniques play a vital role in identifying communities in graphs. The graph decomposition is to divide a graph into a set of dense subgraphs interpreted as communities so that the communities in the graph can be identified and extracted. Then, it is feasible and available to focus on smaller but more crucial areas of the graph when the input graph becomes exceedingly massive. As a definition of dense subgraph, the k-truss formed by triangles is one of the simplest cohesive subgraphs since triangle is the smallest non-trivial clique. By the definition, every edge of the dense subgraph to be decomposed is contained in at least $(k-2)$-triangle. The task of k-truss decomposition is to find all trusses in a graph. The cohesive subgraphs in a graph can be extracted hierarchically based on the k-truss decomposition method. A triangle (or 3-clique), consists of three fully connected nodes. It is the densest substructure in a graph. Therefore, the k-truss keeps a good trade-off between computational efficiency and clique approximation. However, the k-truss is not available for bipartite graphs because no triangle occurs in bipartite graphs. The k-truss decomposition method is not applicable to bipartite graphs.

In this research, by adding additional edges to a bipartite graph, we extend the notion of k-truss to bipartite graphs, name it the *quasi-k-truss*, so that the communities as cohesive

subgraphs can be extracted from a bipartite graph. Then, the original bipartite graph G is transformed to another graph G'. When a k-truss T is computed from G, the quasi-k-truss is the subgraph whose edges are included in both T and G', that is, the graph obtained by removing all additional edges from T. Every edge in the quasi-k-truss is contained in at least $(k-2)$-triangle while a bipartite graph contains no triangles inside. In order to extract triangles virtually in a given bipartite graph, we introduce the auxiliary edges. Alternative to the originally existed edges, the auxiliary edges do not originally exist in bipartite graphs. An auxiliary edge is generated between any two nodes in the same node set if these two nodes share at least one common neighbor node in the other node set. The set of auxiliary edges is generated among all nodes with the same characteristics in a given bipartite graph. The set of auxiliary edges is generated for forming triangles in a given bipartite graph. According to the notion of quasi-k-truss of bipartite graphs, the quasi-k-truss decomposition algorithm is developed for decomposing a bipartite graph, and extracting communities represented by dense subgraphs from a bipartite graph. In the proposed algorithm, initially, a set of auxiliary edges is generated to form triangles in a given bipartite graph. Then, to extract communities from a bipartite graph, the number of triangles containing an edge of the bipartite graph is counted. Communities can be extracted hierarchically from a bipartite graph based on an iterated procedure. In order to improve the computational efficiency, some data structures have been applied to the implementation of the proposed algorithm. Finally, experiments for real world datasets have been conducted to verify the effectiveness and efficiency of the proposed method compared with baseline methods.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Background of Community Extraction

One of the challenging computational problems in graph analysis is dense subgraph detection. Due to the increasingly large size of graphs and the huge amount of data embedded in graphs, both the computational time cost and memory usage become more and more serious. Discovering the important areas of a graph is a helpful and effective method in graph analysis when the given graph is massive. In order to find the important areas of a graph is to identify meaningful dense subgraphs interpreted as communities in a graph. The problem of identifying communities has attracted tremendous attention recently because of the increasing interest in studying various graphs with complicated structures. A graph is considered to hold community structures if its nodes can be clustered together. A community is also called module or cluster in a graph. Numerous algorithms have been developed based on various definitions of community in a graph, such as the *matrix blocking algorithm* for community detection [12], the *stochastic flow based community discovery algorithm* [58], the node degree distribution based *k-core decomposition algorithm* [2] and the triangle based *k-truss decomposition algorithm* [3, 64]. These methods of community extraction are also applicable for analyzing various social networks. Many works of social networks analysis employ the community extraction methods in recent years [6, 32, 48, 55, 63, 65].

The traditional study of sociology is a social science that studies the social behavior, institutions of different nations, organization and development. Various methods of empirical investigation and statistical methods are used to analyze the social order and its organizations, and predict the future development of society critically. Graph plays a crucial role in modeling complicated social relations [46, 62]. In a social network, the prospects of social networks mainly focus on relationships among various social entities. The assumption of importance of relationships among social entities is the basis of social network analysis [18, 63]. The social

networks are, therefore, social structures consist of social actors and various relationships. Social actors represent the set of nodes, and social relationships represent the set of edges or links in a graph. The interest in social network analysis pays tremendous attention to both the relationships among various social entities and the structural patterns of these relationships. The social environment is described as patterns or regularities in relationships among various social entities based on the viewpoint of social network analysis.

In addition, with the development of web technology and the wide spread of Internet, the Internet users and pages also increase rapidly. A virtual platform for building social relations has arisen and became world wide popular corresponding to the traditional social interaction occasions. It is a Web-based social networking service, such as the Facebook, twitter and blogs. The Internet users can register their personal information to website. Then, each user could build the friendship with other Internet users. This is called the *social networking service*, or SNS for short. The diffusion of SNS leads to the creation of virtual (or online) communities on the Web. The group of online users can share their own hobbies, interests, and opinions, or even post their photos to the website. Some professional SNS websites also supply other useful functions to their clients, such as job hunting, or online video lecture for education. SNS websites are varied and the considerably growing size of networks brings a great challenge to the computation and analysis of graphs. The social network online users also incorporate novel information and communication methods, such as blogging, video and photo sharing, mobile connectivity, etc. A huge amount of data and computable resource embedded in the Web-based social networks. Efficient methods for analyzing social networks and mining graph data embedded in social networks become more and more attractive and crucial.

1.2 Community Extraction in Bipartite Graph

Generally, a graph is represented by $G = (V, E)$ with the node set V and the edge set E such that $E \subseteq V_1 \times V_2$. A bipartite graph is a special graph satisfying that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ such that any edge is connecting $u \in V_1$ and $v \in V_2$. Bipartite graphs are often used for modeling the relationship between two classes of objects, such as movies-actors networks, tweets-users networks of twitter, the sales volume-consumers of commercial relationship networks, and the authors-articles networks. However, it is a challenge to analyze bipartite structures and extract dense subgraphs from bipartite graphs due to the special graph structure [8, 21, 37, 43, 59, 61, 68, 69]. Those triangle-based dense subgraph extraction methods are not applicable to bipartite graphs because no triangle exists in bipartite graphs.

A measurement that quantifies the quality of graph division has been proposed by Newman [46]. However, this measurement is available for uniform graphs only. In the previous study [59], the diversity of human interests is concerned for every individual has more than one aspect in the society. Aiming at extracting many-to-many inter-cluster relationship from a bipartite graph. The measurement is extended into a novel modularity measure, such that it becomes applicable to bipartite graphs. The multi-facet communities can be extracted from a bipartite graph based on this extended measurement for bipartite graphs.

A modified clustering coefficient of bipartite graphs is proposed for detecting community structure in bipartite graphs [69]. This is an edge based clustering coefficient given by the fractional cycles in bipartite graphs. The fundamental clique of bipartite graphs is square. Thus, the clustering coefficient of bipartite graphs quantifies the density of squares. In this method, the clustering coefficient of bipartite graphs is defined as the comparison between fractional number of observed squares and the total number of possible squares. Sum up the degree between pairs of neighbor nodes except the node and its neighbor node if they are connected. All of the possible squares can be obtained. The modified clustering coefficient for bipartite graphs considers the summation of degree between any pair of neighbor nodes for the different connections of a node would lead to different characteristics of the node. The edge-based clustering coefficient for bipartite graphs is defined as the number of squares that an edge belongs to. The edge with the lowest value of clustering coefficient is cut. Then, the given bipartite graph is divided by the number of squares that contains the edge. The communities in a bipartite graph can be obtained.

A data clustering method for bipartite graphs is proposed in a study [68]. Cluster analysis aims at partitioning a set of data into clusters such that the set of data belongs to a cluster has higher similarity than those in distinct clusters. In the proposed method, a given bipartite graph is partitioned by a minimized sum of edge weights between any two unmatched nodes. The minimization of edge weights can be obtained by computing the singular value decomposition of the associated edge weight matrix. The quantity of $W(V_1, V_2)$ is used to measure the similarity between the two node sets V_1 and V_2. To partition the set of data into clusters, the similarity between unmatched nodes should be as small as possible such that the bipartite graph can be partitioned. So it is necessary to discover the partitions for minimizing the normalized cut of the bipartite graph. In order to make the connections between two set of nodes to the singular value decomposition problem, the weight matrix is built by the set of similar nodes. The second largest left and right singular vectors of the weight matrix is computed. Then, discover the suitable cut points for the two set of nodes V_1 and V_2 respectively so that the partitions for the two sets of nodes is formed. The dense

subgraphs of the given bipartite graph is recursively partitioned till the densest subgraphs are found.

A hierarchical model is proposed for bipartite graphs [8]. It is based on hierarchical random graph model with associative and dissociative of nodes. The associative behavior is used for clustering nodes. The disassociate behavior is used for dividing nodes from communities into subcommunities. A *Monte Carlo Markov Chain* is employed to generate the optimal bipartite hierarchical random graph model in a given bipartite graph. The hierarchical model space is traversed by using rotation operations on the candidate bipartite dendrograms. The regenerated subgraphs holds the similar characteristics and degree distribution as the observed graph.

Except clustering coefficient for bipartite graphs partitioning, a heuristic algorithm named the *3SHP algorithm* has been proposed for community detection in bipartite graphs [43]. This algorithm consists of three steps to partition a given bipartite graph such that communities can be detected from the subgraphs. As the definition of community in other works, the community is defined as a set of nodes with dense internal connections and sparse external connections among communities. However, the same as other partitioning methods for community detection in bipartite graphs, the difficulty of the 3SHP algorithm is to determine the most suitable cutting point in a given bipartite graph. Finally, the evaluation method proposed by Newman and Girvan is employed for measuring the quality of partitioning. Generally, the 3SHP algorithm consists of three step processings: starting-point identification, preliminary partitioning and the final adjustment. Based on these iterated procedures, a bipartite graph can be divided by selecting two core nodes with high degree, and constructing the community structure from each core node.

The *LP & BRIM algorithm* proposed in [37] is used for detecting community structures in bipartite graphs. This algorithm combines the *label propagation algorithm* for detecting communities, and the *BRIM algorithm* for generating communities by inducing partition recursively between the two sets of nodes in bipartite graphs. The time complexity of *LP & BRIM algorithm* is proved to be no larger than $O(n^2)$.

The *Summarization-Compression Miner algorithm* proposed in [21], also named the *SCMiner algorithm* for short, is to prune the not so important area of the input graph, and reduce the size of the given graph such that highly compact areas of the graph can be focus. The highly compact areas always represent the true information embedded inside the graph. Differ from graph clustering approaches, this technique consists of link prediction, clustering and summarization. It reduces the size of input graph by global abstraction to ensure the features of the input data is distilled and small enough to be accessed. The basic principle of the proposed technique is to transform the input graph into a highly compact summarized

graph. The summarization S of a bipartite graph $G = (V_1 \cup V_2, E)$ consists of two bipartite subgraphs S_a and S_b, a summarized graph G_s and an additional graph G_a. To find out the best summarization of a given bipartite graph, the *SCMiner algorithm* initially merge the nodes sharing similar connection patterns into groups when the similarity between two nodes connected by an edge in the group is larger than a given parameter. The main clusters of both node types and the connection patterns between these clusters can be found.

1.3 The Contributions of This Research

Here we briefly mention the results of this thesis previously presented in [9, 38, 39, 41]. Through the observation on community extraction methods for bipartite graphs, most of them are degree distribution based graph partitioning methods. The method for community extraction in bipartite graphs is few due to the special graph structure of bipartite graphs. Therefore, in this dissertation, we initially examine the method of triangle counting in graphs for dense subgraph extraction as preliminary research because finding the maximal cliques in massive graphs is computationally hard. We also implemented an approximate triangle counting algorithm. Next, we introduce the quasi-truss decomposition algorithm for bipartite graphs by extending the original k-truss algorithm. The k-truss is the largest subgraph in a graph. Every edge of it is contained in at least $(k-2)$ triangles. In here, we should distinguish the k-truss from triangle for k-truss is not triangle. Based on this computation, a graph can be decomposed hierarchically, and all k-truss can be extracted from a given graph. The truss decomposition algorithm avoids the intractable problem of clique detection when extracting a dense substructure community. This method keeps a good trade-off between clique approximation and computational efficiency. The truss decomposition is not, however, applicable to the bipartite graphs due to its definition. To solve this problem, we propose the *quasi-truss decomposition* introducing an additional set of edges, named the *auxiliary edge*. The set of auxiliary edges is proposed for forming triangles in bipartite graphs such that truss-like components can be extracted.

In this research, we firstly develop a algorithm to generate all auxiliary edges in a bipartite graph, such that the triangles in a given bipartite graph are listed. Secondly, we develop one more algorithm to examine the scalability of the initial algorithm for decomposing a bipartite graph based on counting triangles. Finally, we improve the triangle counting based on the quasi-truss decomposition with the purpose of improving the effectiveness and precision of the proposed algorithm. This algorithm is a tailored k-truss decomposition algorithm. We also use several real world datasets in the experiments to verify the practicability and efficiency of the proposed algorithm.

Finally, this dissertation is organized as follow. It is rigorously divided into five chapters. Initially, the chapter 1 introduces the research background about the social network analysis, the motivations for doing this research as well as the related works concerning the triangle listing and counting algorithms. In the chapter 2, a variety of notions of dense subgraphs for community are introduced. Accordingly, numerous techniques for detecting and extracting dense subgraphs are also introduced as the related researches. The proposed algorithms are introduced in the chapter 3. First, we extend the notion of k-truss to bipartite graph, and develop a triangle structure based algorithm called quasi-k-truss decomposition algorithm for bipartite graph. Various data structures are used in implementing these proposed algorithms with the purpose of improving the computational efficiency of these proposed algorithm. A succession of experiments are also done to evaluate the proposed algorithms. The implementation and evaluation of these proposed algorithms are concluded in the chapter 4. In the this chapter, some real world datasets in bipartite structure are used in these experiments. The experimental results proof the practicability of the proposed algorithm. Finally, the chapter 5 concludes the entire research, and discuss the probability of improvement for the proposed technique as the future work.

1.4 Triangle Count: Related Works and Applications

In a graph, triangle represents a smallest non-trivial clique such that three nodes v_1, v_2 and v_3 are fully connected to each other. So triangle is also called clique, and denoted by T where $T_{123} = (v_1, v_2), (v_2, v_3), (v_1, v_3)$. A triangle is a complete subgraph either the edges are directed or undirected.

The Fig. 1.1 shows the structure of triangle. Triangle is one of the fundamental substructures of graphs that it plays an important role in both the computation of clustering coefficient and transitivity analysis of graphs.

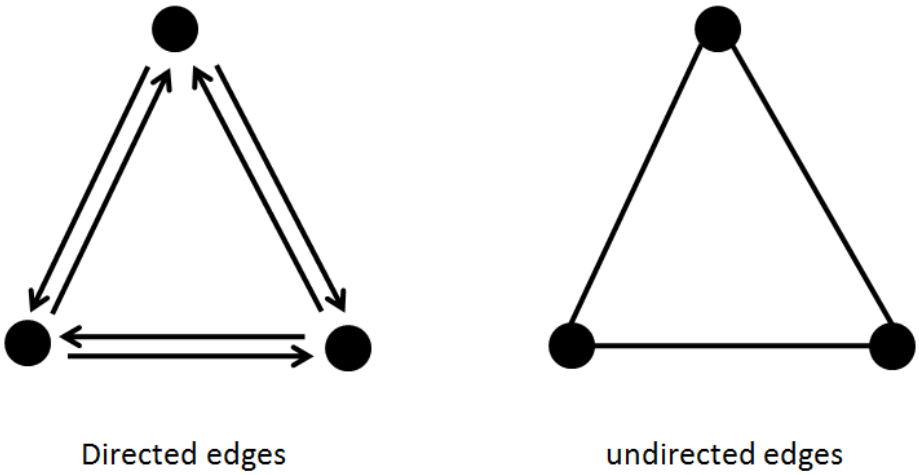

Fig. 1.1 The structure of triangle

1.4.1 Clustering coefficient

The clustering coefficient is a measurement of degree to determine whether the nodes are tend to cluster together and form tightly knit subsets within a relatively high density of connections in a graph or not. Practically, in most of real-world networks, particularly in social networks, nodes tend to construct dense subgraphs in which are characterized by relatively high density of connections, called *small world* [66]. Two versions of the clustering coefficient exist: one is global version, another is local version. The global version gives an overall indication of the clustering in graphs. The local version indicates the connectivity of a single node. A given undirected graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges connecting the set of nodes. An edge $e_{ij} = (v_i, v_j)$ is an edge connects two nodes v_i and v_j. The neighborhood N_{v_i} of the node v_i is defined as its immediately connected neighbors as follow:

$$N_{v_i} = \{v_j | e_{ij} = (v_i, v_j) \in E\}$$

The clustering coefficient of a node is the ratio of number of connections in the neighborhood of a node and the number of connections if the neighborhood is fully connected. Define the degree d_v of a node as the number of nodes of $|N_v|$, then, $d_v = |N_v|$. The degree d_{v_i} of node v_i is the number of nodes connect to v_i but do not include the node v_i itself. Three

nodes build a triangle if two neighbor nodes of a node are connected by an edge. Let λ_{v_i} be the number of triangles including node v_i. A triple at a node v_i is a path of length for which v_i is the central node. Let τ_{v_i} be the number of triples on node v_i which belongs to V. The τ_{v_i} is the number of subgraphs with two edges and three nodes, one of which is v_i and such that v_i is connected to other two nodes by both two edges. Thus, counting the number of triples reveals the number of edges that actually exist for the neighbor nodes of a particular node, and define the clustering coefficient of node ccf_{v_i} as follows:

$$ccf_{v_i} = \frac{\lambda_{v_i}}{\tau_{v_i}}$$

1.4.2 Transitivity analysis

The clustering coefficient of three triples reveals that the transitivity among these triples is defined as the ration between the number of triangles and the number of paths of length in a graph. The computation of transitivity in graphs is based on triangle comparing the relative number of triangles with the total number of connected triple nodes in a graph. In a graph $G = (V, E)$, the degree of node v is defined as the number of nodes connect to node v. A complete subgraph of three nodes of the graph G is considered as a triangle or clique. The number of triangles of node v is defined as:

$$\delta_v = \left\{ \{u, w\} \in E : \{\{v, u\} \cup \{v, w\}\} \in E \right\}$$

A triplet at node v is a path of length two for node v is the central node. The value of transitivity is one when a graph contains all possible edges. Sum up the triple of all nodes, the transitivity of a graph G can be defined as below, where deg_v indicates the degree of node v.

$$\delta_G = \sum_{v \in V} \frac{2\delta_v}{deg_v}$$

Each triangle is counted three times for it participates in other three different connected triples that the number of triangles of a graph. The number of triangles in graph G is defined as follow:

$$\delta_G = \frac{1}{3} \sum_{v \in V} \delta_v$$

From the definition of transitivity of graphs, transitivity is highly related to the clustering coefficient of graphs for both are measurements of the relative frequency of triangles. The

clustering coefficient is used for computing the cluster of nodes such that the density of any subsets in graphs could be examined. The transitivity of graphs is used for traversing the entire graphs and examining the connectivity among nodes in graphs. This characteristic of transitivity can be used for detecting the connectivity occurrence and computing the flow of networks.

1.4.3 Triangle sum for document viewpoint abstraction

An algorithm called the *TriangleSum algorithm* has been proposed for extracting key sentences from a machine readable document [9, 34, 35]. The set of highlighted key sentences summarizes the viewpoint of the input document. Generally, two different approaches for document summarization exist: abstraction and extraction. The abstraction methods summarize the meaning of a source document based on paraphrasing sections. However, the extraction methods merely extract and copy the segmental information considered most crucial to the from a source document.

Initially, the words with high occurrence frequency in source document, such as "the", "a", "is" and "are", are filtered out as meaningless words by using an electronic dictionary. Because these kinds of words are useless in summarizing the viewpoint of the source document.

Any two words w_i and w_j in a sentence s_k are connected based on their co-occurrence relation if the distance between these two words is less than a given threshold. It is denoted as $co(w_i, w_j, s_k)$. The word w_i is said to be the adjacent word of the word w_j in the sentence s_k. Otherwise, these two words w_i and w_j are not co-occurred.

A novel definition named the *dependency frequency* is proposed based on the dependency relationship of the set of remained words to build a dependency graph of words in the source document. A word w_i is considered to be dependent on word w_j in the sentence s_k if word w_i is grammatically modified by word w_j. The dependency relation between any two words w_i and w_j is denoted as $dep(w_i, w_j, s_k)$. A dependency graph is built for all remained words in the source document. All words represent the set of nodes, and both the co-occurrence relation and dependency relation among the words represent the set of edges in the dependency graph.

Given an example document contains two sentences, ⌈*South Korea participates in the joint five-day naval military drill in the Yellow Sea area launched by Navy of United States from Monday to Friday. The Navy of United States sends the nuclear-powered Los-Angels-class submarine to the military activity*⌉.

The meaningless words in the given document are filtered out. A dependency graph is built for the remained words. The Fig. 1.2 illustrates the dependency graph of words remained in the source document. Two triangles anchor in the given document.

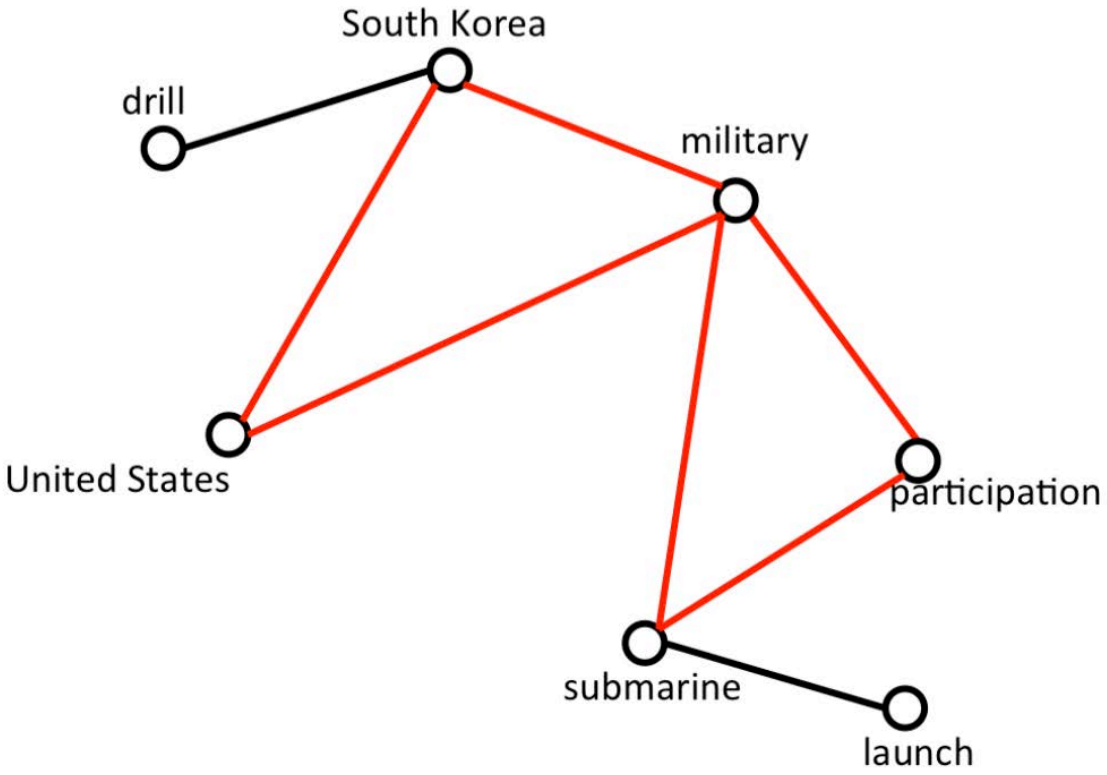

Fig. 1.2 The dependency graph of words

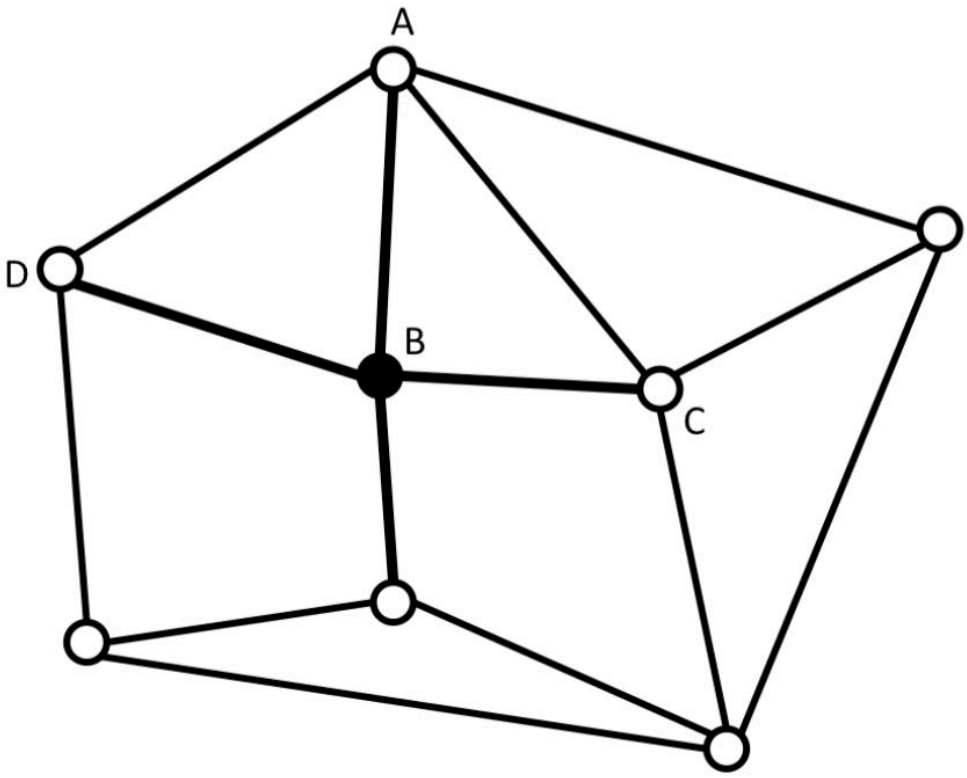

Fig. 1.3 Counting the number of triangles in the dependency graph

Finally, examine the neighbor nodes of a node if they are connected or not in the dependency graph illustrated by the Fig. 1.3. Extract all triangles in which the focused node participates in. For example, the node B is the focused node in the dependency graph. The two neighbor nodes A and C are examined to determine if these two nodes are connected together or not. The node B is said to participate in the triangle $T = \{A, B, C\}$. In addition, other neighbor node D also connects to node A, such that node B is also contained in another triangle $T = \{A, B, D\}$. Counting the number of triangles anchor in every sentence. Sum up the number of triangles, the sentence contains more triangles than other sentences in the document is highlighted to summarize the viewpoint of the document.

1.4.4 Triangle listing in graphs

Triangle listing in massive networks is a basic problem of graph computation [5, 13, 33, 53, 60]. This problem has been researched intensively from a theoretical point of view in the past. With the increasingly large size of graphs, such as the social networks and other types of graphs aforementioned, the efficiency of triangle listing algorithms becomes crucial.

Moreover, the techniques of triangle listing can also be used for community detection, pattern mining and social networks analysis, etc. Triangle listing is to enumerate triangles, and count the exact number of triangles in G. Thus, the methods of triangle listing consider triangle structure as the smallest substructure, and partition a graph into subgraphs when the given graph is too large to be stored and processed in the main memory of computer mainframe at a time. These subgraphs are small enough to be stored in main memory, and processed one by one.

An I/O-efficient triangle listing algorithm has been developed in [5]. The *triangle listing algorithm* is designed for processing the undirected unweighted graphs. It enumerates the exact number of triangles in G, and avoids random disk access. The main principle of the proposed algorithm is to decompose a graph G iteratively, and listing all triangles in each subgraph separately. A subgraph is processed at one time. By using the triangle listing algorithm, a given graph G is decomposed into a set of subgraphs, such that the these subgraphs can fit into the main memory of the computer mainframe. The triangles in each local subgraph will be enumerated. The Fig. 1.4 illustrates the partitioning of a given graph G into subgraphs.

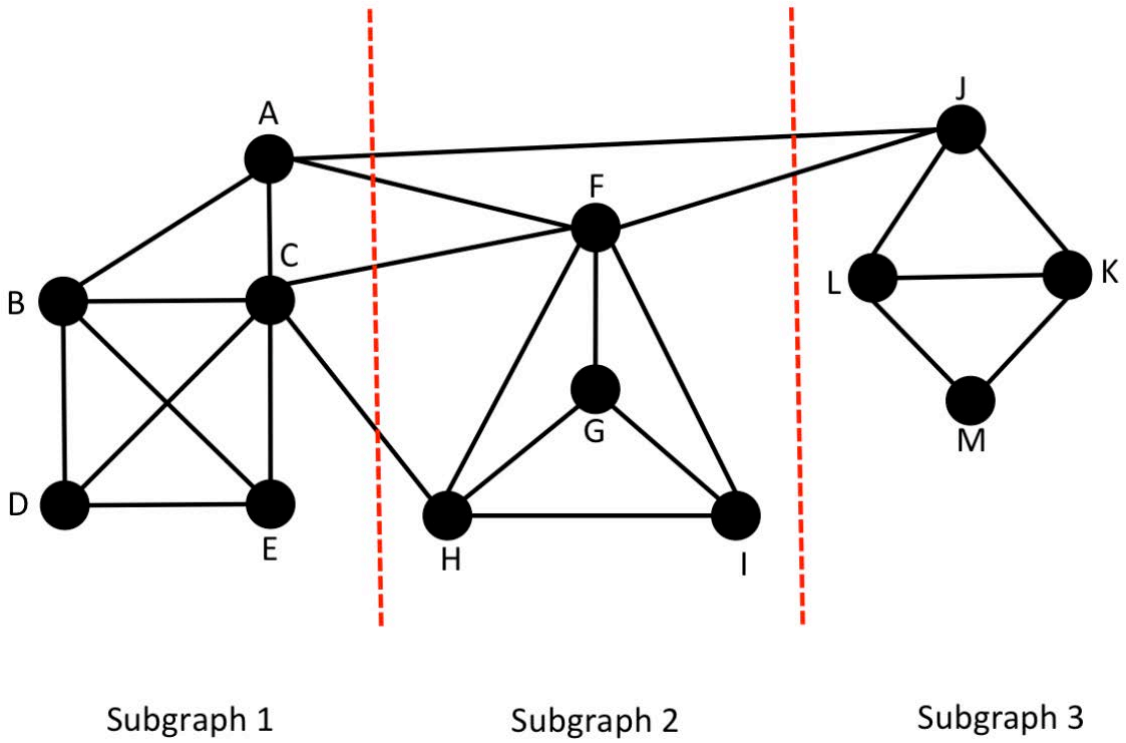

Fig. 1.4 Partitioning of graph with using I/O-efficient algorithm

In the partition processing, the triangles in G will be categorized into three types, *Type 1*, *Type 2*, and *Type 3*, for ensuring the correctness and completeness of the final output result. A triangle is the Type 1 triangle if all its nodes are contained in the same subgraph by the partition. A triangle is the Type 2 triangle if its nodes are contained in two subgraphs respectively. Finally, the nodes of a Type 3 triangle are contained in three different subgraphs separately. In the Fig. 1.4, the triangle T_{ABC}, T_{BCD}, T_{BCE}, T_{BDE}, T_{CDE}, T_{FHG}, T_{FIG}, T_{GHI}, T_{FHI}, T_{JLK} and T_{LKM} are Type 1. The triangle T_{ACF} and T_{CFH} are Type 2, and the triangle T_{AFJ} is Type 3. Mechanically, Type 1 and Type 2 are listed, then, Type 3 triangles are converted into the Type 1 and Type 2 triangles at the next iterative processing. The Fig. 1.5 depicts the extended subgraphs of $1,2,3$ in G after dividing the given graph G. The nodes in black are the nodes in each subgraph. In addition, the nodes in dotted circle line are the extension to nodes outside of each subgraph. Intuitively, an extended subgraph can be obtained by adding edges to connect the nodes inside the extended subgraph with the nodes outside the extended subgraph. The purposes of introducing the notion of extended subgraph are to ensure the completeness of the global result, and to enable the removal of edges after all triangles containing the edges are enumerated.

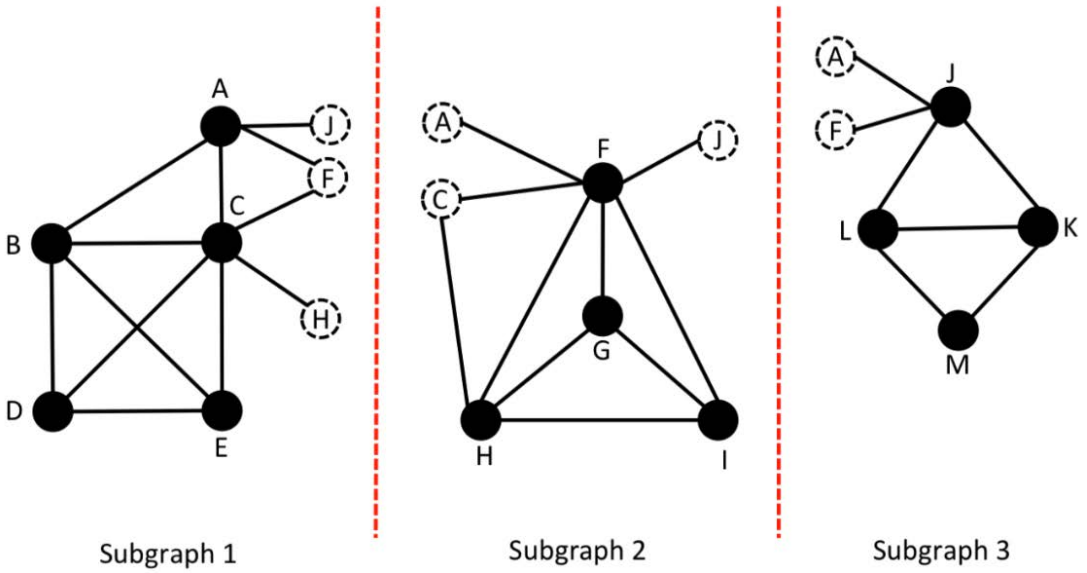

Fig. 1.5 The extended subgraphs

In particular, with the increasing number of nodes and edges in G, the main memory space is not large enough to hold the entire G, such that $(|V|+|E|) > M$, where M stands for the size of the main memory space. Initially, a graph G is partitioned into a group of subgraphs,

$G = s_1, s_2, s_3, \ldots, s_i$. Each subgraph is processed respectively, such that each subgraph s_i can fit into the memory space. Then, loading each subgraph into memory and listing the triangles in the loaded subgraph locally. Finally, those edges of a subgraph s_i that cannot contribute to triangle listing are removed out of G. Repeat these processes iteratively until G becomes empty. The extended subgraph of each subgraph in the partition into memory space is read once at each triangle listing iteration. Therefore, the I/O complexity for one iteration of the triangle listing algorithm is $O(|V|+|E|)$. The theoretical analysis is also provided to prove the correctness of the I/O complexity of the triangle listing algorithm.

1.4.5 Breadth-first search for graph traversal

Initially, the *breadth-first search* is employed to traverse the entire graph G with the purpose of finding all of the triangles in G. The *breadth-first search* is a graph traversal method alternative to the *depth-first search* method. A graph G is systematically traversed. All nodes in the graph are uniformly examined once.

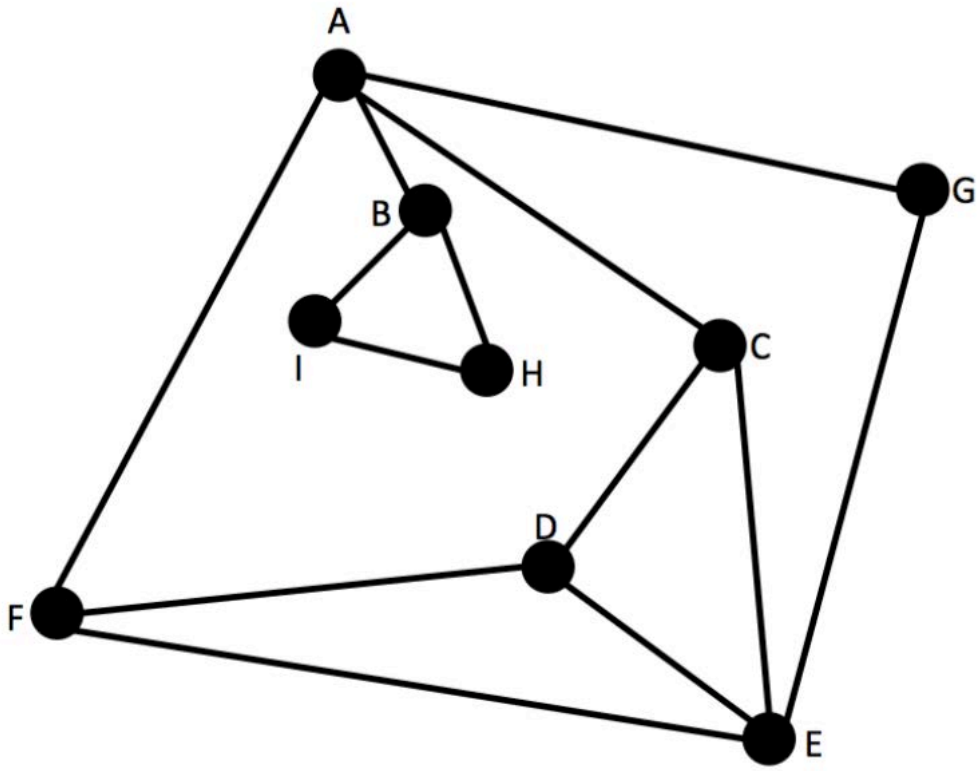

Fig. 1.6 Triangle searching in G

The Fig. 1.6 illustrates the searching path of breadth-first search. Suppose node A is the root node, and search starts from this node. From node A, its neighbor nodes B, C, F, G will be visited. The neighbor nodes B, C, F, G of node A are found. However, there is no triangles anchoring in these nodes for these four neighbor nodes of node A do not connected to each other. Then, the search follows the path to explore the neighbor nodes of node B. Two neighbor nodes H, I of node B are visited. Three nodes B, H, I are fully connected to each other that form a triangle T_{BHI}. There are also two neighbor nodes of node C, nodes D, E. These three nodes C, D, E are also fully connected to each other that form a triangle T_{CDE}. Two neighbor nodes D and E are connected together among three neighbor nodes A, D, E of node F. Therefore, a triangle T_{DEF} can be found. The search stops until all neighbor nodes of the last node are found in G. Three triangles T_{BHI}, T_{CDE} and T_{DEF} are found after traversing the entire graph G.

1.4.6 Exact and approximate triangle counting

Differ from triangle listing algorithm, the techniques of triangle counting proposed in [13, 53, 60] count the exact or approximate number of triangles. The *EIGEN-TRIANGLE algorithm* is proposed for counting the total number of triangles [13]. The idea of the EIGEN-TRIANGLE algorithm can be summarized as follows: the number of paths of length 3 that begin and end at the same node i is contained in the diagonal element α_{ii} of the adjacency matrix A. The adjacency matrix is then converted to a square matrix A^3. This implies that the node i is contained in a triangle. In addition, a triangle consists of three nodes that all triangles in G are triple counted while traversing the entire graph. Thus, the trace of the square matrix A^3 is three times of the total number of triangles. The total number of triangles in G is one sixth of the sum of cubes of eigenvalues.

There are two usages of the eigenvalue in this algorithm: can be computed for sparse graphs and applied on the mainframe of hadoop. Moreover, the top-i-th eigenvalue λ^i is an eigenvalue of the square matrix A^i, where $i \geq 1$, if λ is the eigenvalue of the adjacency matrix A. A succession of experiments has been done to count the number of global triangles and the number of local triangles, where the global triangles are represented by the triangles in G, and the local triangles are represented by these triangles in which a certain node participates in. From the theorems and the idea of the proposed algorithm, the G will not be partitioned into subgraphs when counting triangles in G. The triangles in G is also unnecessary to be enumerated as the output result.

With the increasingly large size of graph, finding the maximal triangles in a massive graph is computationally hard. Alternative to the exact triangle counting algorithms, another type of algorithms proposed for counting the approximate number of triangles in G with

certain given parameters or threshold are randomized algorithms [53, 60]. The idea of these algorithms are similar to the sampling approaches that correlate the sampling of node or edge in G with parameter.

To count the approximate number of triangles, every node in graph G is given with a unique color. The edge is called monochromatic edge if two nodes of it receive the same color. The set of monochromatic edges is sampled. In a triangle of graph G, is counted if three edges of it are monochromatic edges. The third node or edge of a triangle is sampled if the other two nodes or edges of a triangle are sampled. This is the main idea of the *coloful triangle counting algorithm* [60]. The set of edges is sampled with a probability. Define N as the number of given colors that each edge could be sampled with a probability p, where $N = \frac{1}{p}$. The more number of colors are given to nodes, the lower probability of edges to be sampled.

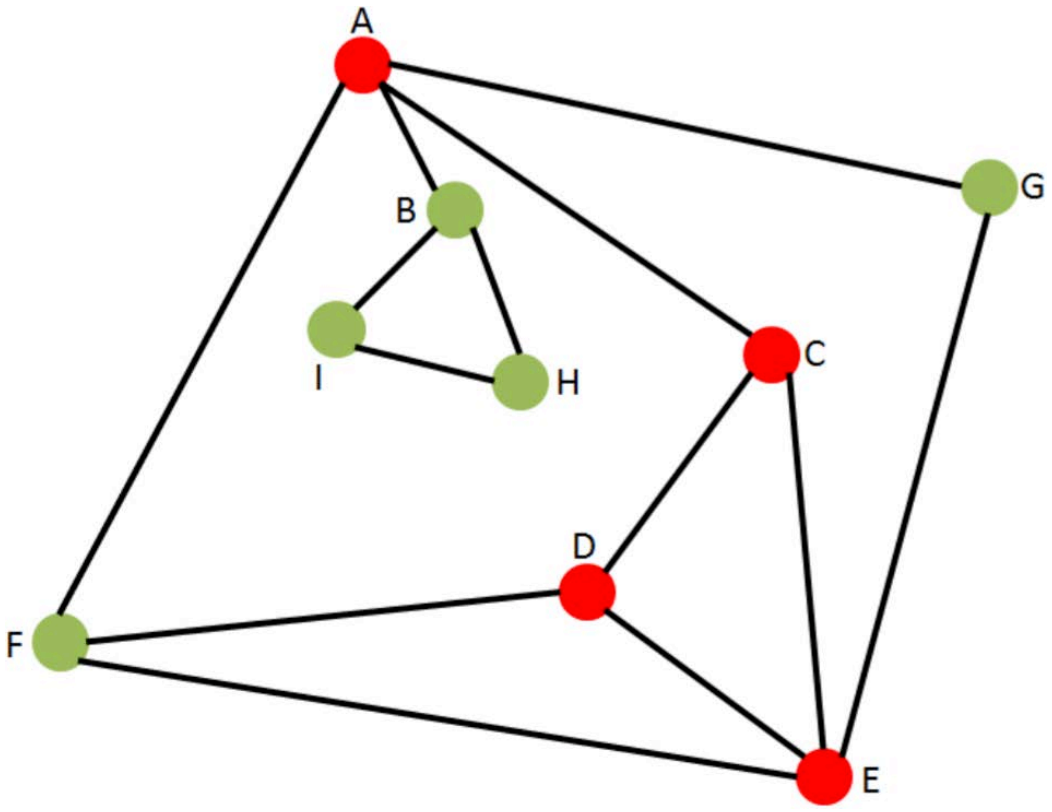

Fig. 1.7 Approximate triangle counting in graph

The Fig. 1.7 illustrates the principle of colorful triangle counting algorithm. From root node A, its neighbor nodes B, C, F, G are visited. Then, each node is given with a unique color at the same time. The node A and node C are colored with red. The node B, node F and node G are colored with green. However, there is no triangles anchoring in these nodes.

The search follows the path to explore the neighbor nodes of node B. Two neighbor nodes H, I of node B are found. The two nodes H and I receive the green color, the same color with the color of node B. Thus, the triangle T_{BHI} is counted as a triangle in G for its nodes are colored with green. On the other side, the node C is also shared by two nodes D and node E. The node D and node E connect to each other. Then, the triangle T_{CDE} in G is found. Shown as the Fig. 1.7, the two nodes D and E receive the same color as the node C. The triangle T_{CDE} satisfies the parameter of the triangle counting algorithm. The triangle T_{CDE} is counted. However, node F receives different color from the two nodes D and E although the three nodes D, E, F construct a triangle T_{DEF} in G. The triangle T_{DEF} is not counted for its two edges $e = (D, F)$ and $e = (E, F)$ are not sampled. The number of triangles estimates the exact number of triangles in G.

1.4.7 Experimental results of approximate triangle counting

We implemented the colorful triangle counting algorithm in [42]. The Table 1.1 below shows the features of the datasets. The *web-Google* dataset records the webpages and the hyperlinks which is represented by a set of directed edges. The *roadNet-CA* dataset records the transportation network of California.

Table 1.1 Features of datasets

dataset	$\|V\|$	$\|E\|$	type	size(in byte)
web-Google	875,713	5,105,039	directed	96,259,277
roadNet-CA	1,965,206	5,533,214	undirected	87,765,811

The experimental results of web-Google and roadNet-CA were shown as Table 1.2 and Table 1.3. The N indicates the number of given colors. The computation time is counted in the unit of second. The number of T indicates the number of triangles. The size indicates the size of sampled graphs, which is defined as $(|V| + |E|)$. The experimental results in the Table 1.2 and Table 1.3 can be concluded in two aspects. First, the colorful triangle counting algorithm is available for both directed graphs and undirected graphs. Second, with the increasing number of given color, the counted number of triangles and the computation time decrease.

Table 1.2 Results of web-Google

N	time(in sec.)	# T	size(in byte)
10	5137.18	230,018	8,036,948
20	739.92	64,738	4,309,952
30	259.43	32,317	3,003,158
40	112.67	19,181	2,300,624
50	65.92	12,302	1,861,096
60	42.95	9,264	1,570,576
70	32.67	7,670	1,370,426
80	24.05	6,473	1,225,008
90	19.80	5,332	1,092,404
100	16.75	4,738	1,004,500

Table 1.3 Results of roadNet-CA

N	time(in sec.)	# T	size(in byte)
10	1529.02	8,632	10,003,648
20	329.91	2,600	5,304,008
30	140.37	1,568	3,600,260
40	87.83	1,032	2,732,824
50	48.07	880	2,191,022
60	33.53	760	1,830,712
70	25.19	712	1,585,094
80	19.60	676	1,391,908
90	18.32	648	1,244,074
100	13.26	627	1,123,620

Another approximate triangle counting algorithm called *DOULION triangle counting algorithm* is a significantly useful and applicable algorithm in all kind of scenarios no matter the size of the given graph G fits into the main memory space of the computer mainframe or not [60]. In this algorithm, a coin is tossed by DOULION for every edge. Similar to the colorful triangle counting, every edge is sampled with a probability. The probability reduces to $1-p$ when the edge is deleted. Then, every triangle in the remained graph G' is counted as $\frac{1}{p^3}$ triangles. After tossing the coin for each edge, in the next stage, the NODEITERATOR is used as the triangle counting black box in the mainframe of the DOULION triangle counting. It counts the number of neighbor nodes of each node to determine the number of edges exist among its neighbor nodes. The value of probability p ranges from *0.1* to *0.9* within nearly 100% accuracy no matter the input graph is large size or small size.

Chapter 2

Community Extraction from Graphs

2.1 Community of Graphs

The literal meaning of community itself contains social background context. The local community could be a group of residents lives in the same building, or a group of students studies in the same class. The global communities could be economic unions among countries, or humanitarian organizations and other non-governmental organizations around the world. A pragmatic example of graph with communities is the social networks. A community is also called a cluster or module that a group of nodes have common properties. Many community exist in most of the real world networks. A network is considered to have the community structure if its nodes can be clustered into subsets with common characteristics, such as degree [19, 22, 25, 46, 52]. The definition of a community in networks implies that the community is represented by dense subgraphs in the graphs. The community structure is one of the common properties of graphs [19, 22, 25, 52]. Each subset have strong internal connections among all of its nodes. All subsets are, however, weakly connected externally in the graph. This property reveals that a graph can be naturally divided into dense subgraphs with weak connections among these subgraphs. Usually, communities may overlap with each other in graphs.

The Fig. 2.1 shows the community structure in a graph example. In the given graph, some nodes of it connect to the main structure of the graph weakly, or even totally isolated from the main structure of the graph. Oppositely, some nodes connect to each other densely, even potentially overlapping. These subsets of nodes within dense internal connection are considered as the communities in the given graph. The inhomogeneous connection signifies the characteristic of certain natural divisions of graphs in both symmetrical structure and asymmetric structure. Community is defined in various types of forms, and represented by dense subgraphs in graphs. Numerous definitions of the dense subgraph have been

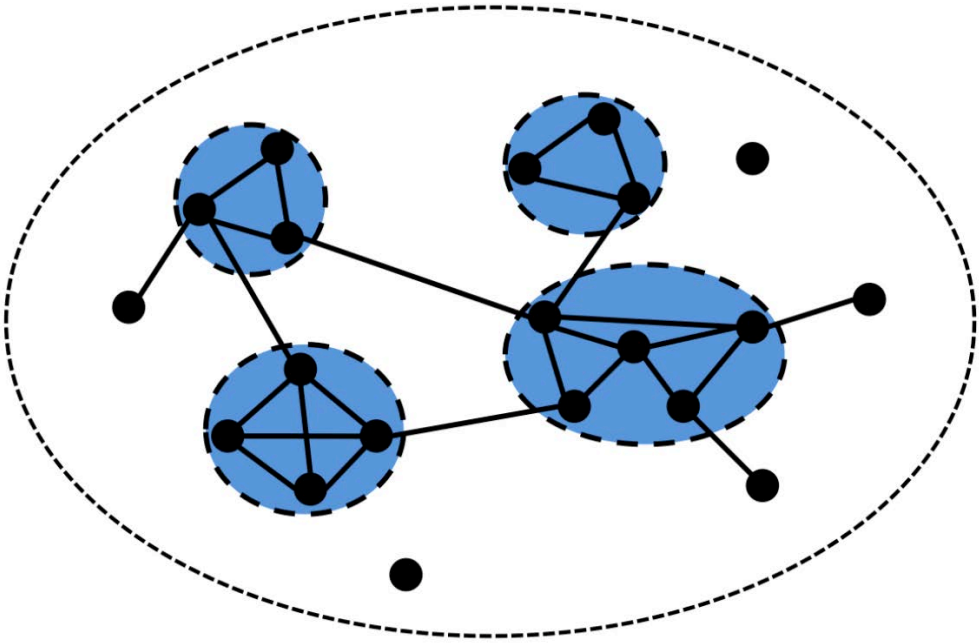

Fig. 2.1 The community structure in graphs

proposed [19, 22, 52]. Community structure is one of the graph infrastructures that useful in analyzing the hierarchical organizations of networks. Moreover, identifying communities in graphs is crucial in detecting the modules and their boundaries in graphs. This allows for clustering nodes based on their structural positions in the modules.

2.2 Techniques of Community Extraction

A Community in an arbitrary graph is usually defined in terms of partitioning the set of nodes and edges. The task of community extraction refers to find out the subsets of nodes which are more strongly connected internally than other part of the graph. The community extraction has been a serious concern in the graph module and hierarchical structure analysis. Detecting community in an arbitrary graph is, however, NP-hard. The number of communities, if any, within a graph is normally unknown. The size and density of communities are usually unequal. Being able to discover the communities in graphs can provide insight into the complicated affect between the function and topological structure of graphs [19]. Although various techniques for community extraction have been proposed in the past decade, the effective evaluation for these techniques and measurements for extracted communities are still remained difficult. In this section, some definitions of communities of graphs will be

introduced, and the corresponding community extraction techniques will also be described in detail.

2.2.1 Graph partitioning

The community detection has been a focus of some research fields with diverse applications. Computationally, it is an NP-hard problem in the study of graphs due to the complicated graph structures. A variety of methods of community extraction have been proposed in the past decade [20, 30, 36, 50, 67]. A classical method of community extraction is to partition an entire graph into several subsets of nodes in which represented by dense subgraphs in the graph. A partition is a division of graphs in modules or clusters, such that nodes are clustered into several modules within predefined size. The number of edges that lying among the subsets of nodes is minimal. The process of partitioning is called *cut size*.

A community is defined as the tightest subgraph in which contains maximal number of edges among a set of nodes [67]. Therefore, the proposed algorithm focuses on the edges in a candidate community and the edges that connect the candidate community to the rest of the graph. Otherwise, the edges will not be focused on if they do not belong to the focused candidate community. This framework extracts a densest community from a given graph at one time. The remained graph is allowed to mix the cohesive and sparse communities with weakly connected components. Accordingly, a given graph can be cut into two parts. The candidate community and the rest of the graph are asymmetric. Given a graph $G = (V, E)$, where $|V|$ is the number of nodes and $|E|$ is the number of edges in G. Define an $|V| \times |V|$ adjacency matrix for representing the connectivity occurrence among all nodes in V, denote it as $M = [M_{ij}]$. The connectivity occurs between node i and j if $M_{ij} > 0$. Otherwise, there is no connection between node i and j if $M_{ij} = 0$.

The Fig. 2.2 illustrates the principle of graph partitioning. The graph G is partitioned into two communities COM_A and COM_B, where $COM_A \cup COM_B = V$ and $COM_A \cap COM_B = \Phi$. The red dashed lines illustrate the boundary of graph partition between two communities. The community COM_B has denser internal connections than the community COM_A obviously. Thus, the community COM_B is considered as the candidate community, and extracted. The partitioning between two communities of unequal size with minimal number of edges connecting the two communities. A community that has maximal number of edges itself and minimal number of edges to the remained graph is extracted at one time. In here, an important point is needed to be stated that the partitioning of graphs in this community extraction algorithm is different from the graph partitioning of triangle listing algorithm. The structure of the extracted community is not extended. The sparse edges among communities in G are removed such that one community could be extracted at one time.

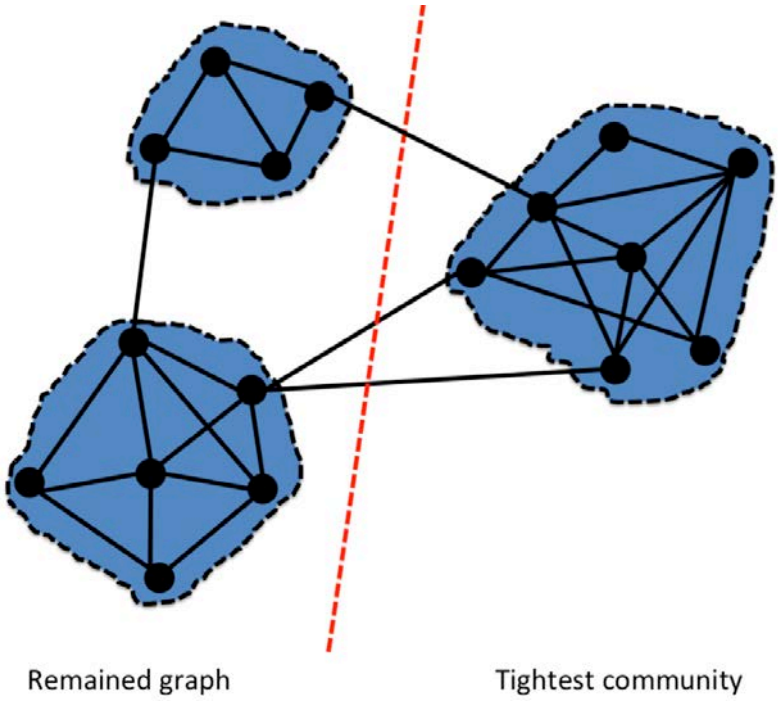

Fig. 2.2 Graph partitioning

The candidate community and the remained graph are not symmetry. Then, the extraction criterion is maximized with all possible cases. The total number of possible connections normalizes edges in the internal community and that to the rest of the graph G in each case. A natural division is given to all edges in G as probabilistic interpretation. The proposed extraction criterion is not only fits the intuitive definitions of community, but also has a natural interpretation within a wide class of probability models on graphs. To maximize the extraction criteria, tabu search is used as a local optimization method in this proposed algorithm. All nodes $v \in V$ of G is visited and marked as a tabu node. The algorithm begins from an initial value. The label of nodes are switched. The nodes are marked as tabu nodes. Then, the algorithm returns to the first node when the given initial value of the global maximum is improved. The node that enlarge increase within the criterion value is switched. The global maximum cannot be improved till no nodes can be switched. The algorithm runs with a certain number of iterative processes. Some social network datasets are used in the experiments to prove efficiency of the proposed technique in both tight community extraction and probabilistic interpretation.

With the wide spread of Internet and the fast increase number of Internet users around the world, another virtual platform of social networking based on World Wide Web has formed. The *facebook* and *twitter* are world wide famous social networking service providers. They

can be considered as social networks of an virtual world, encoded inside the Internet. In recent years, the SNS has attracted tremendous attention. Initially, the World Wide Web was created for exchanging information among a group of people internally. However, with the rapid development of web technology and the fast growing of Internet users, a variety of web based business and applications have already been developed. Nowadays, most of people gets into the habit of surfing the Internet to browse the website for searching their required information, or for entertainment occasionally. An estimated statistic about the total number of web pages is more than 25 billion on the World Wide Web [46]. Therefore, a novel and crucial Internet tool has been constructed for crawling web pages and retrieving information based on the input keyword from the Internet user. The text of web pages related to the input keyword will be discovered and listed automatically so that the Internet user can find out the needed information on the World Wide Web.

Technically, the website can be written by the Hypertext Markup Language to specify the appearance, and transmitted over the Internet based on the Hypertext Transport Protocol. The World Wide Web is obviously a graph of information and data for the web page consists of hypertext files, pictures and other kinds of information in specific formats. Considering the World Wide Web as another huge graph corresponding to the Internet, the web pages represents the set of nodes in the web graph, and the nodes connected by the hyperlink which represents the set of edges in the web graph. The hyperlinks allow people to navigate the web pages from one to another. The web graph is a directed graph for the hyperlink between any two web pages points at one web page from a starting page. The community structure also exists in the web graph according to its topological linking structure. They could be related to a group of websites dealing with similar topics, or pointing at the same direction to a root web page. Detecting the community structure in the web graph is helpful in extracting useful information from the websites [16, 22]. Furthermore, it is also useful in identifying the artificial modules generated by the hyperlink farms with the purpose of reinforcing the PageRank value of websites and endowing them with a higher ranking. Correspondingly, a crawler has been built to measure the structure of the web graph. The crawler plays a fundamental role in retrieving the websites. Starting from any root website, the text of that website will be downloaded. Then, all of the hyperlinks in the text can be found. The hyperlink consists of the identification tag and the Uniform Resource Locater, or URL for short functionally. The identification tag is a short text that used to mark the hyperlink as a link of website. The Uniform Resource Locater is a standardized computer address that tells the Internet user where and how to find out the linked website. The crawler can extracts the URL for all hyperlinks from a website, and stores the extracted URL in the memory of computer mainframe after scanning the identification tag and copying the adjacent URL. The

website is removed from the storage when the process for the current website has completely done. These processing procedures will be iterated.

To identify the communities in the web graph, various efficient methods have been proposed in [11, 14, 22, 23, 47, 54]. Communities in the web graph are sets of websites represented by nodes in G. Each member of a community has more edges connects to other members than to non-members [22, 23]. A community of the web graph is defined as a node-based dense subgraph with strong connections internally. According to this definition, communities that represented by dense subgraphs within more edges internally than other parts of the web graph, such that it can be identified by partitioning the entire web graph. The weight of edges between any two partitioned communities is minimized. Besides, the edge directions are removed that communities in G are undirected subgraphs. This is the ideal definition of community on the web graph in [22, 23].

In this algorithm, the mainframe of *Hyperlikn-Induced Topic Search* [24, 29, 49] where HITS for short, is employed for crawling the websites. The Hyperlink-Induced Topic Search is a link analysis technique that used in search engine, such as *Google* and *Yahoo!*. The HITS algorithm is used for searching the most relevant websites according to the query with a certain insight into the creation of webpages. The group of certain websites is called the *hub*. The hub connects to other authorities, and serves as directories that leads Internet users to other authoritative websites. Generally, a hub website links to many authority webpages, and an authority webpage is linked by many hub webpages.

The Fig. 2.3 shows the relationship between *hubs* and *authorities*. Therefore, the hub and authority are defined for recursion and strengthen with each other mutually. Formally, the HITS algorithm searches the set of seed websites that relevant to the input query mostly at first. Next, a set of basic websites is generated by the set of seed websites for including most of the strongest authority websites as its second step. The set of basic websites and all of the hyperlinks among the connected webpages form a subgraph of websites embeds in the Internet. This kind of website subgraphs is focused on. Due to the characteristics of HITS algorithm, it is employed as the webpages crawler for the seed website is necessary to the proposed technique as the starting point. However, on the other side, the members of a community could belong to other communities, and communities would be overlapped with each other in such a case. Moreover, to identify communities from the small world networks without dominate members is difficult by using the HITS algorithm.

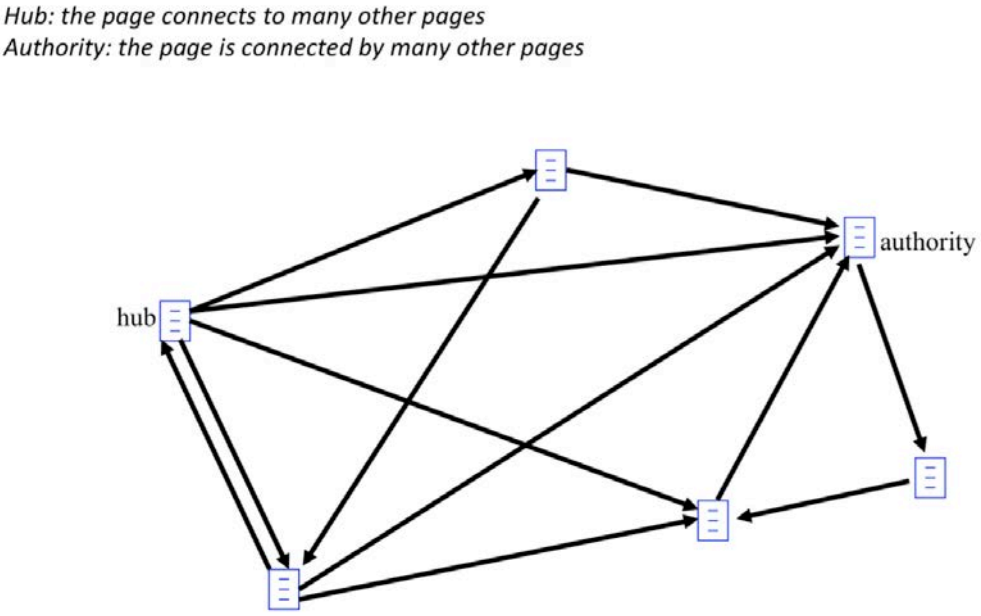

Fig. 2.3 The relationship of hubs and authorities

In the proposed method, the *HITS algorithm* is applied for crawling websites. Then, two quantities are defined for characterizing the adequacy of the source and sink because the edges are allowed to be weighted. The communities with weighted edges in either direction can be identified in G by using the *maximum flow/minimum cut* technique between the source nodes and the sink nodes. First, the source is denoted as s, and the sink is denoted as t. A community is denoted as C, and the set of its nodes is denoted as V. Second, the *source link count* is defined as the number of edges between source s and all of the nodes included in $(C-s)$, denoted as $s^\sharp$. The *sink link count* is defined as the number of edges between sink t and the set of nodes in $(V-C-t)$, denoted as $t^\sharp$ in [22]. Use the minimum cut between the nodes of source s and the nodes of sink t, where $s-t$ in G, the community C can be identified. The nodes are in the identified community C if they can be reached from the nodes of sink s after the cut. The proof is also summarized and provided. The defined community can be identified in G. Besides, the parameters of inbound and outbound for approximation of communities are also considered in the proposed algorithm. The crawler begins retrieving with the seed webpages. Then, all webpages that connect to or connected by the seed webpages are found from the set of seed webpages. The outbound connections are found by inspecting the HTML of webpages. The inbound connections are found trivially querying the modifier of a search engine that supports the connections. A small subset of a

community can be identified with a small number of seed webpages. Because the out degree of the seed webpages has to be larger than the removed number of links.

As an improved web community identification method, the hyperlinks among websites are considered as the basic block of the community in [23]. The web graph has the characteristic of self-organization for there is no authority or process manages the structure and formation of hyperlinks. The structure of web graph is asymmetric and self-organized. Accordingly, the web community consists of a set of websites that each website of it has more hyperlinks than the websites in the outside of it. Moreover, all nodes of a community have more than 50% of their hyperlinks contained inside the community. The communities can be identified by varying the size and the density. The membership among communities is defined as a function based on the inbound and outbound connections of hyperlinks among all websites, such that the communities are formed naturally. Therefore, the edge-based web communities highlight the highly related websites. The improved algorithm uses the information of hyperlinks only. The crawling process can be simplified. Additionally, as the previous algorithm, the maximum flow / minimum cut method is also employed. The framework of maximum flow is used for analyzing the flow or weight of edge between any two nodes in G. The maximum flow is identical to the minimum cut due to the MaxFlow-MinCut theorem. The *exact-flow-community algorithm* includes three processing steps. First, a source node is added with infinite capacity edge, and connected to all seed nodes. Each preexisted edge is bidirectionalized and rescaled to a constant value. Second, a residual-flow graph is generated by applying the maximum-flow procedure. Third, the source node s accesses all nodes through the non-zero positive edges to identify the desired communities from the web graph. The experimental results also proof the efficiency of the proposed algorithms in identifying communities from the web graph.

2.2.2 Clique detection

Communities are not only strongly interconnected, but also potentially overlapped with each other in graphs, such as sharing nodes and edges among communities. In social network of the real world, an individual holds the membership of a hobby group and medical association at the same time. We can say the individual is the intersection of the hobby group and medical association. The hobby group and medical association are overlapped. Therefore, communities are said to be overlapped with each other if two or more communities share at least one node.

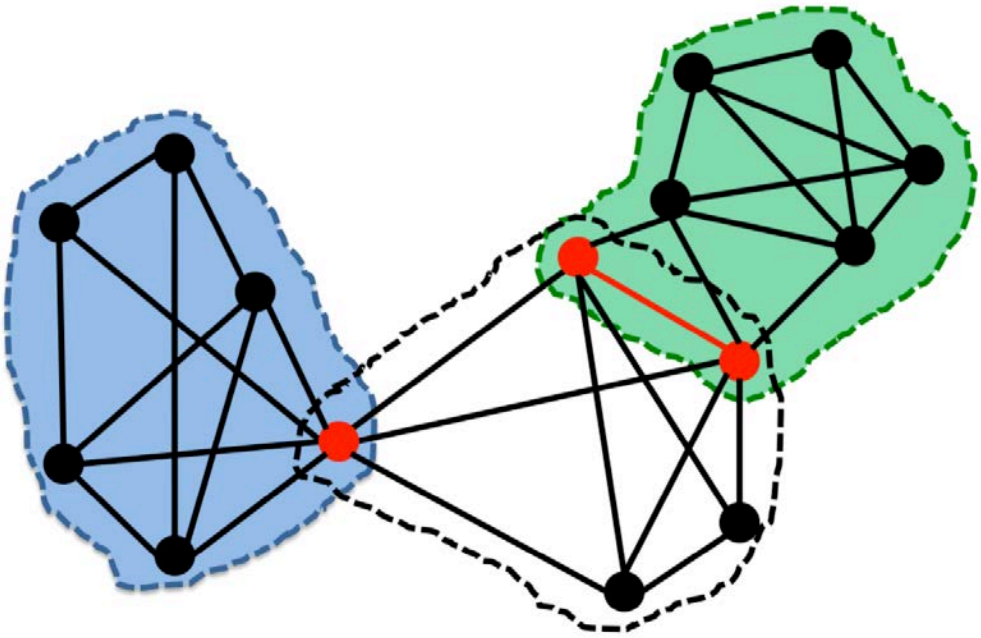

Fig. 2.4 Clique percolation and overlapping

The Fig. 2.4 shows the overlapping among communities in a graph. Two red nodes that are the endpoints of a red edge shared by two communities in graph G. In addition, a red node is also shared by two communities. Two communities are said to be overlapped with each other if at least a node i belongs to two communities COM_A and COM_B, described as $\{i : i \in COM_A \text{ and } i \in COM_B\}$. Then, $i \in COM_A \cap COM_B$.

The graph partitioning methods search for the separated communities in G, but somehow, the communities could be highly overlapped with each other, such as the situations illustrated by the Fig. 2.4. A community identification method is proposed for analyzing the main statistical features of overlapped communities in [50]. In this technique, the nodes in G are labeled with certain numbers indicate their membership of communities. The membership number is the number of communities in which the node belongs to. Any two communities C_A and C_B in G can share n number of nodes, denoted as $n^{ov}_{A,B}$. Therefore, a node could belong to more than one community in G. This is defined as the *overlap size* of two communities C_A and C_B. The size of any community in G is defined as the number of its nodes. Additionally, the overlapped edges of community C_A in graph G are defined as the degree of community C_A, denoted as d^C_A. The community of G is constructed by a set of *k-clique* complete subgraphs within k size where they can be reached from each other by sharing $k-1$ nodes mutually [17, 50]. According to the notion of *k-clique-community*, all

nodes of a unique community is reachable through well connected nodes internally. However, it is not reachable from a specific k-clique substructure of a unique community to other parts of G. Conceptually, the definition of community employed in this technique satisfies five basic requirements: to be localized, to be overlapped, based on the density of edges, neither cut-node nor cut-edge is produced, and not too restrictive. It is a polynomial problem to determine the full set of k-clique-communities in G according to the requirements.

An efficient algorithm for identifying such communities has been developed. Initially, all of the cliques will be located in G. Next, a standard clique-clique overlap matrix analysis technique is executed for identifying the communities in G. Besides, the value of threshold k can be considered as a restriction of resolution. The communities become smaller, denser and more disintegrated when value of threshold k increases. The change of community structure can be observed through some ranges of value of threshold k. In this algorithm, the binary graphs with undirected and unweighted edges are used corresponding to the actual data. The graph G is cut into many pieces as small as possible. Each node of G is forced to remain in a unique community, and separated form other communities, such that most parts of the G fall to pieces and vanish. A succession of experiments has done to proof the efficiency of the k-clique-community identification technique. Three real world network, the network of word associations, the social network of scientific collaborators and the protein-protein interactions network in biological science [50], have been chosen and used as the experimental datasets. In these experiments, the size of community s^{com}, the degree of community d^{com}, the overlap size of community s^{ov}, and the membership number of node m as four cumulative distribution characteristics have been observed.

2.2.3 The k-core decomposition

Communities are represented by dense subgraph in G with various form of definitions. A well-known dense subgraph called k-core is node degree based dense subgraph in [2, 7, 15, 45]. The notion of k-core was proposed by Seidman in 1983. A graph G consists of a set of nodes V, and a set of edges E, where $G = (V, E)$ in general. A subgraph of G is maximum subgraph if a set of nodes $v \in V$ of the subgraph has a degree $deg_v \geq k$, and the set of nodes $v \in V$ forms a core of the subgraph in G. The core within the maximum degree is considered as the main core in G [2, 15, 45]. The k-core is computed by pruning all of the nodes with their degree smaller than k. Their respective edges are also removed out of G. This means a node i becomes $deg_i - n$, and be pruned when $k \geq (deg_i - n)$ if a node i has degree deg_i, and it has n neighbor nodes with degree lower than k. The nodes are removed out of G recursively if its degree is lower than k. Then, a given graph G is decomposed into subgraphs within smaller size, which are easier to be computed. A brief summary for the definition of k-core,

a k-core is a subgraph that the degree of all its nodes is higher than or equal to k. Nodes are considered as coreness k if they do not belong to $(k+1)$-core in G. A k-core can be obtained by removing the nodes which have degree lower than k. The Fig. 2.5 illustrates the principle of k-core decomposition intuitively.

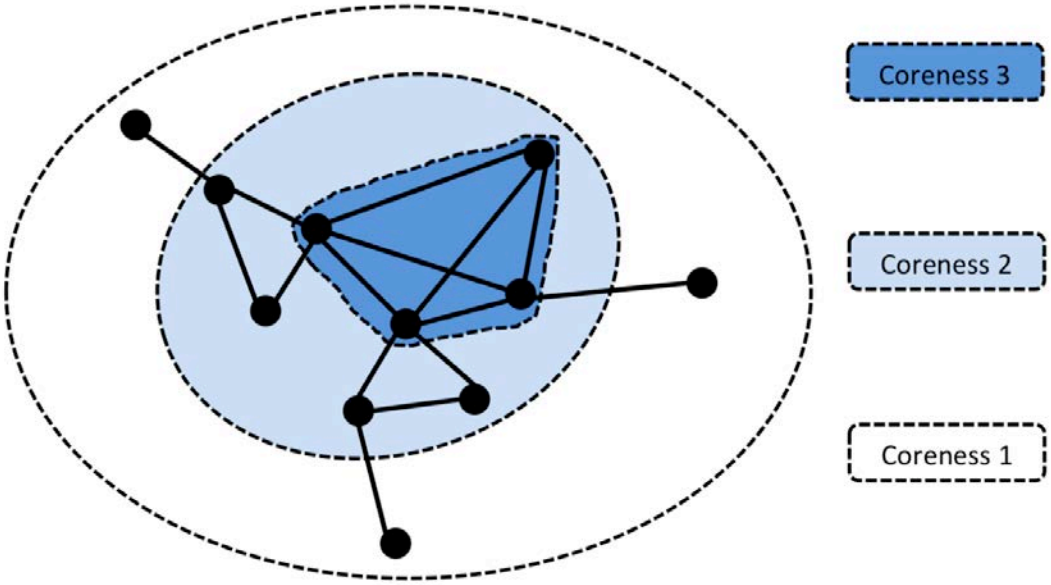

Fig. 2.5 The k-core decomposition

A centralized algorithm for core decomposition based on the notion of k-core has been proposed in [2]. It has been concluded and well described. Give a graph G, the output by using the proposed algorithm is a set of nodes with the core number. For the implementation of the *k-core decomposition algorithm*, array is used to store all of the nodes in G. The array and some local variables are initialized. Next, the degree of each node in G is computed and recorded in the array *deg*. The nodes are sorted within the increasing order of their degrees from 0 to the maximum degree in the array *bin*. The array bin 0 begins from position 1 where each node has a specific position in the array *pos* according to its degree, such that the starting positions of bins in the array can be determined. It is easy to find out which bin each node belongs to and what the starting position of that bin is. The array bin records the possible degree of the first node of its position in the array pos. The node can be fixed into a correct location in the array, and its position is recorded in the array pos. The nodes of G are sorted based on their degrees when increasing the starting position of bins. The values

in array bin for a position are moved to the right for restoring the correct starting positions. The core number of current node is the current degree of the current node. This number is recorded in the array *deg*. The degree of each neighbor node of a node should be decreased, and moved for one bin to the life side in the array bin if it has a higher degree. This process can be done within a constant time. The entire algorithm runs in time $O(\max(m, n))$, where m indicates the number of edges, and the n indicates the number of nodes in G. The running time is $O(\max(m, n)) = O(m)$ for $m \geq (n-1)$ in a connected graph. The analysis of the time complexity of the k-core decomposition algorithm has been described in detail.

Some other properties of the k-core decomposition are also well studies in [7, 15, 45] for its various applications, such as clustering in complex networks, characterizing social networks, distributed system analysis and graph visualization. A distributed version of k-core decomposition is also considered. Accordingly, a distributed algorithm for k-core decomposition is proposed in [45]. This is motivated by the limited memory of computer mainframe that a graph is too large to fit into the memory, and the inherent distribution over a group of nodes causes the inconvenience movement of each portion to a centralized position. The distributed k-core decomposition algorithm is a locality-based k-core decomposition because of the maximality of cores in G. The coreness of a node has at least k neighbor nodes within the k-core or larger core if the node has the largest value of k. This property is formalized as a theorem in [45]. Two computational models are considered in this algorithm: the one-to-one and the one-to-many. The one-to-one model focuses on a node which it associates with a computational unit, and connectivity occurs among nodes through direct communication. On the other side, a host collects many nodes and their edges together when the connectivity occurs between hosts in the one-to-many model.

The procedure of the proposed algorithm is summarized briefly in this paragraph. A node $i \in V$ produces an estimate with the information of its coreness, and send the information to its neighbor nodes as a message. The message includes its identifier and degree, denoted as $\langle i, d(i) \rangle$. The process is called communication intuitively. The initial value of estimate is equal to the degree of a node. Each node receives estimates from its neighbor nodes, and uses these information to recompute its estimate. In the case of changed which represents a Boolean flag, a node sends its new value of estimate to its neighbor nodes, and the process continues till converged. In the case of the one-to-many model, a host is responsible for a group of nodes. The host runs the distributed k-core algorithm for its nodes. The values of estimates are collected and stored, then, sent as messages to other hosts which are responsible for their neighbor nodes. The algorithm can be optimized by updating the estimate when a node receives a message $\langle j, k \rangle$ where $k < est[j]$. Here, the $est[]$ is the variable integer array. The integer array contains an element for each neighbor node. All of its entries is initialized to

$+\infty$ without more exact information. A host receives the messages of estimates from outside that new value of estimates can be generated for some of its own nodes. Therefore, other estimates can be generated among all nodes of the host by these process. All of the nodes in the host becomes quiescent when there is no new internal estimate generated. According to this process, all new estimates are produced and sent to the adjacent hosts. The proposed algorithm is proved to be bounded by $1 + \sum[d(i) - k(i)]$, where $i \in V$. Its computational time cost increases linearly with the size of the given graph G. There is also a further study of k-core decomposition on random graphs in [15]. Different random graph models generate different probability distributions on graphs. Therefore, the studies of k-core decomposition on random graphs address on the degree distribution of nodes in G.

2.2.4 The k-truss decomposition

Triangle is one of the smallest complete subgraph for nodes cannot be more tightly connected than this. All nodes of a triangle are fully connected to each other. There are various approaches to community extraction in graphs based on detecting triangles in graphs. A friend's friend of mine could be my friend. Then, these three social actors are represented by nodes, and edges among the three nodes represent the relationship in the social network. To simplify this network structure, it is a triangle. In the real world, two strongly connected social actors would connect to more than one social actors, and form a social entity group. Based on this observation about such a social structure, an attractive notion of dense subgraph named k-truss has been proposed in [3, 64]. It is a relaxation of clique. Describe the notion of k-truss intuitively by using social network as an example, two people A and B are connected together by a specific social relationship, and there are four more people each link to the two people A and B. The links connecting actors A and B participate in 4 triangles if the link is considered as an edge $e = (A, B)$ in the subgraph. This implies the subgraph consists of 6 nodes and 9 edges. The notion of truss is defined by such triangles embedded in a graph. It is a non-trivial, one-component subgraph that every edge is contained in at least $(k-2)$-triangle. This is defined as the support of an edge $e = (u, v) \in E_G$, denoted by $\sup(e)$. Thus, the support of an edge e in G is the number of triangles in G that contain the edge e. The k-truss is a type of cohesive subgraphs represents a maximal subgraph in a graph G, but not triangles in a graph. The k-truss of G where $k \geq 2$, denoted as T_k so that for all $e \in E_{T_k}$, $\sup(e, T_k) \geq (k-2)$. The task of truss decomposition is to find all trusses in a graph G where $2 \leq k \leq k_{max}$. The truss number of an edge e in G is defined as $\max\{k : e \in E_{T_k}\}$, denoted by $\phi(e)$. Alternative to the definition of truss number, another definition k-class denoted by Φ_k, is defined as $\{e : e \in E_G, \phi(e) = k\}$.

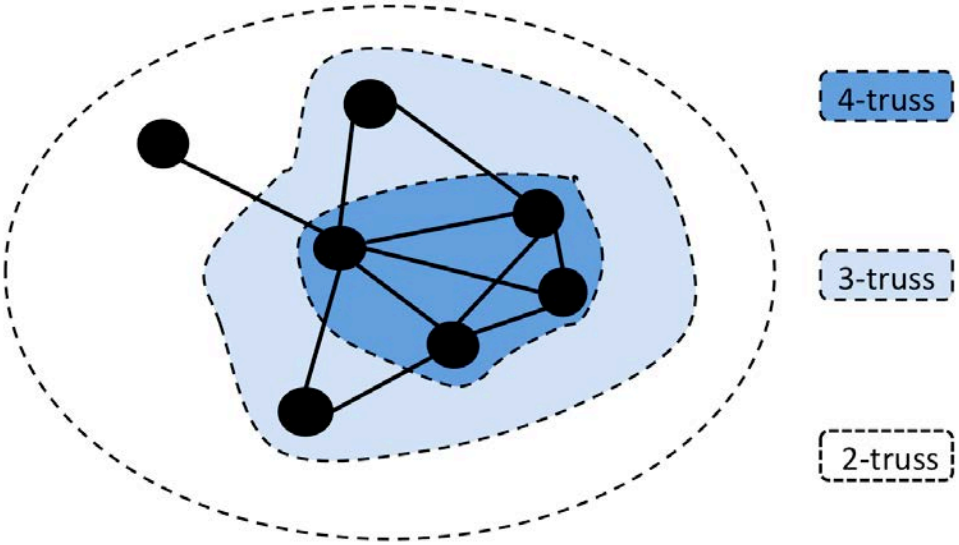

Fig. 2.6 Illustration of the 2-, 3-, and 4-truss decomposition

The Fig. 2.6 illustrates the k-truss decomposition of a given graph G. The edges are contained in different number of triangles in G. The 2-class Φ_2 is the set of edges e with $\sup(e) = 0$. The 3-class Φ_3 is the set of edges with $\sup(e) = 1$, i.e., for $e = (x, y)$, there exist at least one node z such that $(x, z), (y, z) \in E_G$. The 4-class is analogous. From the k-classes, k-trusses of G can be obtained. The 2-truss subgraph is the graph G itself. The 3-truss subgraph consists of the set of edges in $\Phi_3 \cup \Phi_4 \cup \Phi_5$, etc. Each edge of k-truss is contained in at least $(k-2)$-triangle where $2 \leq k \leq 5$. The trusses in different level of granularity represent the hierarchical structures of G. The task of truss decomposition is to find all trusses of G for all K. Most of the social network analysis techniques seek to identify dense subgraphs in G, such as the k-truss cohesive subgraph. Therefore, the k-truss decomposition technique has developed for mining all truss components in a given graph by decomposing the G into a set of small cohesive subgraphs.

To compute the most cohesive subgraphs is NP-hard. Moreover, most of the existed algorithms for computing k-truss are inefficient to handle the exceeding massive graph data nowadays. Thus, an in-memory algorithm has been improved for computing the k-truss within moderate size graph [64]. The complexity of the improved in-memory truss decomposition algorithm is the same as the worst-case of the lower-bound complexity of in-memory triangle listing algorithm. Additionally, two I/O-efficient algorithms have been

proposed for handling graphs that cannot fit into the main memory space of the computer mainframe. One is a bottom-up approach that uses an effective pruning strategy by deleting a large portion of edges before the computation of every truss. Both the I/O cost and the consumption of retrieval space are significantly reduced based on this procedure. Another is a top-down approach that the k-trusses of larger values of k are preferred, as they represent the backbone or core of graphs.

The improved in-memory algorithm is initialized with computing the support of edge for all edges in G. The initialization processing can be done within $O(m^{1.5})$ computation time by the in-memory triangle counting algorithms [33, 57]. Then, all of the edges is sorted based on their ascending order of support. These sorted edges are stored in an array. The sorting process can be done within $O(m)$ computation time and $O(m)$ memory space by employing the bin sort for no edges are added or deleted. After the initialization, the support of edges is computed for every edge. The intersection of the neighbor nodes of any two nodes (a, b) connected by an edge $e = (a, b)$ build triangles within the two nodes (a, b). The number of neighbor nodes of node a and b is equal to the number of triangles containing the edge e_{ab}. The cardinality of the intersection is the support of edge, denoted by $\sup(e)$. Each k begins from $k = 3$. Those edges are removed out of G if their support of edge are less than $(k-2)$. However, a triangle no longer a valid triangle in the graph when one of its edge has been removed. The support of other two edges in a triangle is decreased. The processing of edge removal is repeated iteratively until all of the remained edges in G have the support of edge at least $(k-2)$. These remained edges construct the k-truss. The algorithm has been outlined. The implementation of the improved in-memory algorithm has also been highlighted in the paper [64]. This algorithm requires $O(m+n)$ memory space within $O(m^{1.5})$ computation time to compute the k-truss for all $k \geq 3$. Therefore, the larger graph of size $(|V| + |E|)$ requires the larger memory space. This is expensive for computing massive networks with nodes of high degree.

Besides, two I/O-efficient truss decomposition algorithms have been presented for reducing the I/O cost when the input graph size cannot fit into the main memory space. One method is the bottom-up approach that begins from the smallest value of k, where $k \geq 2$. The lower bound on the truss number of edges is determined for extracting a candidate subgraph. The candidate subgraph contains all edges in the k-class, denoted by Φ_k. Then, computes the k-class Φ_k, removes all of the edges in Φ_k from G. This is an in memory processing. It proceeds to compute the Φ_{k+1}, and remove the edges in Φ_{k+1} till there is no edges in Φ_k for the removal processing. Another method is the top-down approach that starts from the largest possible value of k. An upper bound on the truss number of edges is determined for extracting a candidate subgraph. The candidate subgraph contains all edges in which belong

to Φ_k. Then, computes the Φ_k in main memory, and removes the unimportant edges. Then, the algorithm proceeds to compute the Φ_{k-1}. The processing is repeated till there is no edges for removal.

The bottom-up approach decomposes graphs from a lower level within the small value of k, such that the extracted subgraphs could be as small as possible. Furthermore, it is more efficient in deleting the irrelevant edges for computing the substructures of k-classes than the top-down approach. On the other hand, the top-down approach is particularly suitable in discovering the trusses within as high value of k as possible, but less efficiency in computing all k-trusses in all level. The experimental results prove that the k-truss decomposition algorithm extracts denser subgraphs than the subgraphs extracted by the k-core decomposition algorithm.

2.3 Classification of dense subgraphs

Various notions of dense subgraphs have been proposed for improving the efficiency and precision of computation, and the density of the extracted subgraphs as well. There is conclusion of dense subgraph category in [10]. Many applicable methods have been proposed according to these notions of dense subgraphs. We summarize all notions of dense subgraphs in the category table of the Table 2.1 with their relevant algorithms and the time complexity.

Table 2.1 Variation of dense subgraphs and related algorithms

Name	Algorithm	Time complexity
Clique	Coppersmith-Winograd algorithm	$O(n^{2.376})$
k-clan	state-of-the-art algorithm	$O(nm)$
k-club	$O^*(1.62^n)$ exact algorithm	$O(n^3)$
k-clique	combinatorial algorithm	$O(n^{\frac{k-1}{2}})$
k-clique community	Kernighan-Lin algorithm	$O(n^3)$
k-component	Expectation-Maximization algorithm	$O(kn^2)$
k-plex	branch-and-cut algorithm	$O(n^4)$
strong LS-set	Girvan-Newman algorithm	$O(mn^2)$
LS-set	non-polynomial algorithm	$O(n^4)$
lambda-set	maximum flow / minimum cut algorithm	$O(n^4)$
weak LS-set	Girvan-Newman algorithm	$O(mn^2)$
k-truss or k-dense	k-truss decomposition algorithm	$O(m^{1.5})$
k-core	k-core decomposition algorithm	$O(m)$

Chapter 3

Quasi-k-truss Decomposition for Bipartite Graphs

3.1 The Method of Quasi-k-truss Extraction

The structure of bipartite graphs differs from other ordinary graphs. It is worth of studying the features of bipartite graphs. In this research, we mainly focus on analyzing bipartite graphs with purpose of extracting quasi-k-truss substructures from bipartite graphs. However, the k-truss decomposition algorithm is not applicable to bipartite graphs due to the special graph structure. Therefore, firstly, we extend the notion of k-truss to bipartite graphs. A notion of auxiliary edges $e' \in E'$ is proposed for triangle formation in bipartite graphs. Alternative to the set of edges $e \in E$, the set of auxiliary edges $e' \in E'$ does not originally exist in bipartite graphs. Then, a triangle listing algorithm is developed for generating all auxiliary edges, and enumerating all triangles in a given bipartite graph. Secondly, we develop another algorithm called quasi-k-truss decomposition algorithm for decomposing a bipartite graph based on triangle counting procedure. Finally, we improved the proposed quasi-k-truss decomposition algorithm for extracting the truss-like components which represent communities in a given bipartite graph. We also employ the density of graph to evaluate the quality of extracted communities and the efficiency of the proposed algorithms [38–41].

3.1.1 The structure of bipartite graphs

Given a bipartite graph $G = (V_1 \cup V_2, E)$, where V_1 and V_2 indicate the two sets of nodes, E indicates the set of edges. The set of nodes of a bipartite graph is divided into two parallel node sets V_1 and V_2. The set of edges $e \in E$ connects a node in V_1 to another node in V_2. However, the nodes in the same node set do not connect to each other. Therefore, triangles

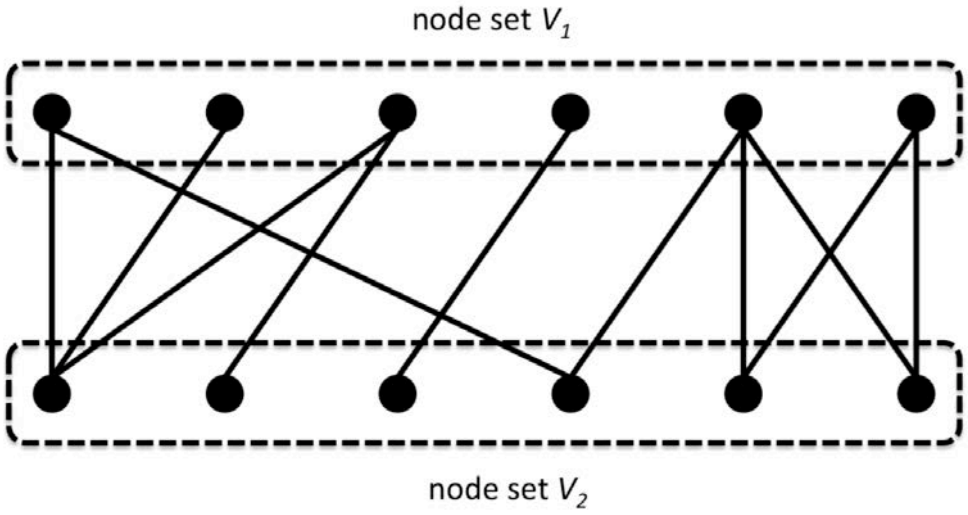

Fig. 3.1 The structure of bipartite graph

do not exist in bipartite graphs. The Fig. 3.1 illustrates the structure of a bipartite graph. From the Fig. 3.1, bipartite graphs contain no triangles. This characteristic of bipartite graphs limits the effectiveness and efficiency of bipartite graphs analysis methods.

3.1.2 The formation of triangles in bipartite graphs

In this work, we proposed the auxiliary edge $e' \in E'$ such that $E' = \{(u,v) \mid u,v \in V_1, u \neq v$, and $x \in V_2$is adjacent to $u,v\}$. The auxiliary edge $e' \in E'$ connects any two nodes in the same node set of a bipartite graph. For two distinct adjacent edges $e_1, e_2 \in E$ in G, there exists a triangle with an auxiliary edge $e' \in E'$. A given bipartite graph G can be transformed into $G' = (V_1 \cup V_2, E \cup E')$ by generating the auxiliary edges. The set of auxiliary edges is generated for constructing triangles in bipartite graphs. Therefore, the number of triangles in a given bipartite graph does not determined by the number of auxiliary edges in the bipartite graph, but determined by the connectivity occurrence between the two sets of nodes. The truss-like component in a given bipartite graph can be extracted based on computing the number of triangles in every hierarchy.

The Fig. 3.2 illustrates the principle of generation of auxiliary edge e' in a given bipartite graph G. Two nodes v_2 and v_3 are contained in the same node set, where $v_2 \cup v_3 \in V_1$. The

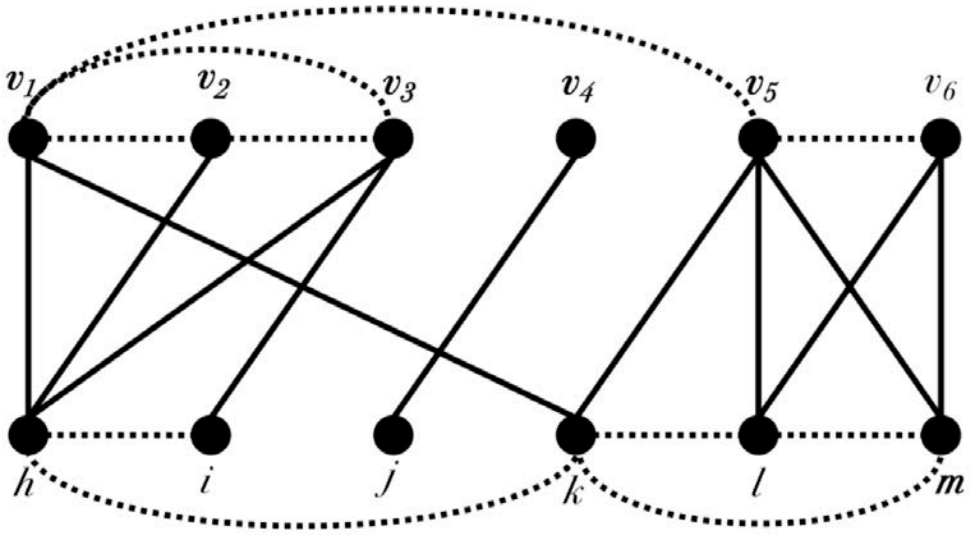

Fig. 3.2 Generation of auxiliary edges in bipartite graph

two nodes v_2 and v_3 share a common neighbor node h in another node set, where $h \in V_2$. Then, an auxiliary edge e' is generated to connect the two nodes v_2 and v_3, such that a triangle $T_{23h} = \{(2,3),(3,h),(2,h)\}$ is formed in the given bipartite graph G. The auxiliary edge e'_{23} is, therefore, contained in another triangle T_{23h}. On the contrary, an auxiliary edge is not generated between the two nodes if these two nodes do not share any common neighbor nodes in another node set. Summarize the characteristic of the auxiliary edge $e' \in E'$, an auxiliary edge e' is generated to connect any two nodes in the same node set of G if these two nodes share at least one common neighbor node in another node set. Otherwise, the auxiliary edge e' is not generated. In the transformed bipartite graph G', two types of edges are contained in the bipartite graph: the original existed edges $e \in E$ which is in solid line, and the auxiliary edges $e' \in E'$ which are illustrated by the dotted line. With generating auxiliary edge $e' \in E'$ in a given bipartite graph G, triangles are formed, and the given bipartite graph G is transformed into G' where $G' = V_1 \cup V_2, E \cup E'$.

3.1.3 The extraction of quasi-k-truss

Conceptually, the notion of truss-like component in bipartite graphs is similar to k-truss, named quasi-k-truss, or quasi-truss for short. It is the densest subgraph represented by

community in bipartite graphs. In the quasi-truss, every edge of it is contained in at least $(k-2)$-triangles. To extract the quasi-truss from G', it is essential to count the number of triangles containing a certain edge $e \in E$. The edges $e \in E$ are recursively removed out of the G' until the densest subgraphs in G' are found. According to the definition of quasi-truss, the dense subgraphs consist of edges $e \in E$. The set of auxiliary edges $e' \in E'$ are excluded, but for constructing triangles in the bipartite graph only.

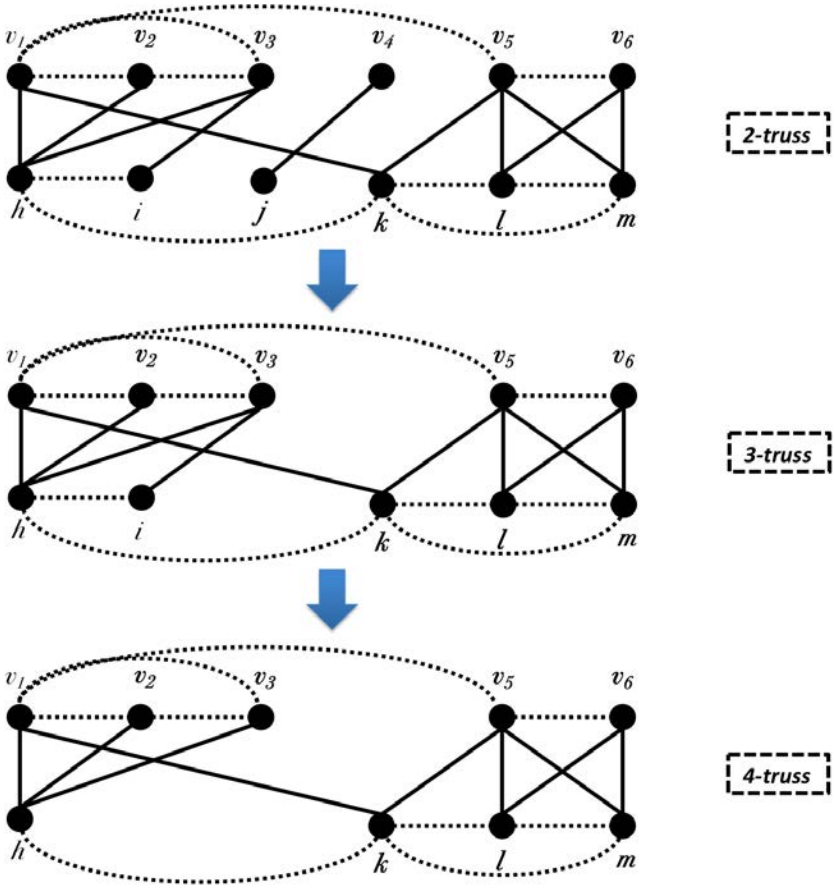

Fig. 3.3 The extraction of quasi-k-truss from bipartite graph

The Fig. 3.3 illustrates the extraction of quasi-truss in each hierarchy. In the given bipartite graph, the 2-truss subgraph is the original bipartite graph itself for the dense subgraph contains all of the edges e, including the edges e are not contained in any triangles, such as the edge $e_{v_4 j}$. The 3-truss dense subgraph includes the set of edges e which are contained in at least one triangle. The 4-truss dense subgraph includes the set of edges e which are contained in no less than two triangles. The 4-truss dense subgraph contains the set of edges $\{(v_1,h),(v_2,h),(v_3,h),(v_1,k),(v_5,k),(v_5,l),(v_5,m),(v_6,l),(v_6,m)\}$. In the given bipartite graph, the densest subgraph is the 4-truss subgraph for the removal of any adjacent edges of edge e leads to the decrease of the number of triangles containing edge e. The process is repeated to decompose the bipartite graph and extract the trusses hierarchically.

3.2 The Quasi-k-Truss Decomposition Algorithms

Corresponding to the conceptual proposal, the relevant algorithms are also developed. These algorithms will be introduced in this section.

3.2.1 Triangle listing in bipartite graph

The first one algorithm for triangle listing in bipartite graphs is developed because the computation of quasi-truss is supported by the triangle listing algorithm. The algorithm is outlined in algorithm 1. For the implementation of the triangle listing algorithm in bipartite graphs, array is employed to store the edges e in memory. Array is a type of data structure consists of a set of values or variables. Each of these elements is identified by at least an array index. An array is stored in the memory of the computer mainframe, such that the position of each value or variable can be computed based on its index.

Initially, an array A of length $2n$ is built in order to store every pair of nodes which represent edges $e \in E$. Every two nodes in the same row of A will be matched whether these two nodes have common neighbor node in another row of A or not. The auxiliary edge e' is generated to connect the two nodes in the same row of A, and output these three nodes as a set of triple nodes represents a triangle. Ultimately, enumerate all triangles containing the set of auxiliary edges e' after matching all of nodes in G.

Algorithm 1: *Triangle Listing Algorithm for Bipartite Graphs*

- q := queue for graph traversal
- E' := the set of auxiliary edge e'
- T := a set of triangles in G

Input: $G = (V_1 \cup V_2, E)$
Output: T

```
1 begin
    init T = φ
    for v ∈ (V1 ∪ V2) do
        v.mark = 0
        q.enqueue(v0)
        while not q.empty() do
            v = q.dequeue()
            v.mark = 1
            if v ∈ e(v, vj) ∩ e(v, vk) then
                e' = (vj, vk) generated
            end
            T = T ∪ {v, vj, vk}
        end
        all triangles in T
    end
    return
  end
```

3.2.2 Triangle counting based bipartite graph decomposition

The quasi-truss decomposition algorithm is developed based on the triangle listing algorithm for bipartite graphs. This algorithm is efficient in decomposing bipartite graphs based on counting the number of triangles with a given parameter. The algorithm is summarized as algorithm 2. In this algorithm, we employ the hash table to store and sort the set of edges $e \in E$ and the set of auxiliary edges $e' \in E'$ in the quasi-truss decomposition algorithm. As a data structure that efficiency in graph searching, especially tree structure searching and other table structure searching, hash table is used to implement an associative array which can map indexes to values. Ideally, a hash table employs the hash function to compute an index into the buckets or slots in an array, such that the correct value can be discovered.

In this algorithm, initialize the hash table of E, denoted as hash[E], and the triangle set, denoted as T. The graph traversal begins from node v. An auxiliary edge e'_{jk} is generated to connect any two nodes v_j and v_k directly connect to v. Then, $T = \{v, v_j, v_k\}$ can be formed in G' after this process. The two edges $e = (v, v_j)$ and $e = (v, v_k)$, and the auxiliary edge e'_{jk} are contained in a triangle. All of the edges $e \in E$ and auxiliary edge $e' \in E'$ are stored in the hash[E]. For any two nodes in the same node set connected by an auxiliary edge e', the number of their common neighbor nodes is equivalent to the number of triangles contains the auxiliary edge e'. The hash[E] stores the set of edge e connecting the two endpoints of an auxiliary edge e' and their common neighbor nodes in another node set instead of storing all triangles in an array [39]. The memory usage can be significantly reduced based on this process. To extract the quasi-truss that represents the largest community, it is essential to count the total number of triangles containing the set of edge e in hash[E]. A parameter is given to the algorithm as a threshold to delete the edges e which are contained in less number of triangles than the given parameter. These edges e do not satisfy the given parameter will be removed out of the hash[E] when counting the number of triangles. Finally, enumerate all triangles which satisfy the given parameter. The given bipartite graph is decomposed by enumerating the set of triangles at different level of granularity.

Algorithm 2: *Quasi-k-truss Decomposition Algorithm*

- q := queue for graph traverse
- Q := input parameter for the number of triangles of an edge e
- E := the set of edge e
- hash[E] := the hash table of E
- T := a set of triangles in G
- $num_{(T_e)}$:= the number of triangles containing an edge e

Input: $G = (V_1 \cup V_2, E)$, Q = 1,2,3,4, . . . , m
Output: T with parameter Q

```
1  begin
       init hash[E] = φ, T = φ
       for v ∈ (V1 ∪ V2) do
           v.mark = 0
           q.enqueue(v0)
           while not q.empty() do
               v = q.dequeue()
               v.mark = 1
               if v ∈ e(v, vj) ∩ e(v, vk) then
                   e' = (vj, vk) generated
                   T = T ∪ {v, vj, vk}
               end
               hash[E] = hash[E] ∪ hash[e'jk]
           end
           for Q = 1 to m do
               for e ∈ hash[E] do
                   if num(Te) < Q then
                       remove e from hash[E]
                   end
               end
           end
           a set of T contains edge e
       end
       return
   end
```

3.2.3 The improvement of quasi-k-truss decomposition algorithm

However, the triangle counting based quasi-k-truss decomposition algorithm has drawbacks in extracting quasi-truss components from bipartite graphs. It focuses on counting the number of triangles in bipartite graphs so that a given bipartite graph can be decomposed. Therefore, the extracted communities contain odd-cycles. It is difficult to measure the extracted communities, and evaluate the effectiveness of the proposed algorithm.

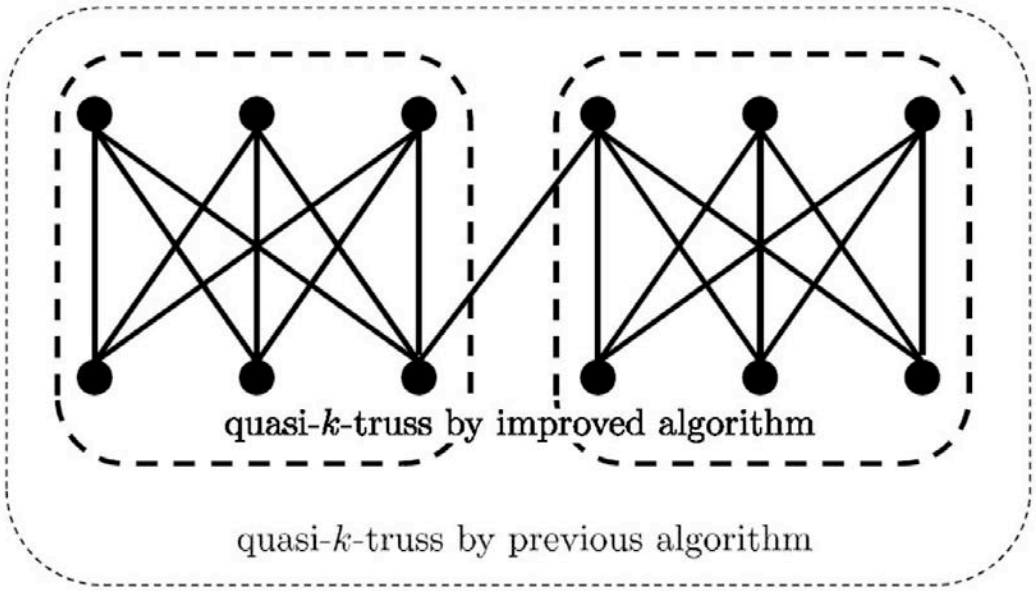

Fig. 3.4 An example of quasi-k-truss detected by the improved algorithm

Besides, consider the situation that two complete subgraphs G_1 and G_2 in bipartite graph, they are connected by an edge shown as the Fig. 3.4. The previous algorithm extracts the structure $E(G_1) \cup E(G_2) \cup \{e\}$ because e is detected as an edge of higher quasi-k-truss than other edges in G_1 or G_2. The reason is that e is the most frequent edge appearing in triangles. In the improved algorithm, we propose a method for removing those undesired edge e based on the appearance frequency of edge e in triangles, such that the two dense subgraphs could be extracted. Otherwise, the edge e with highest appearance frequency in triangles and its adjacent edges are extracted. To avoid such a case, we improve the quasi-k-truss decomposition algorithm described in algorithm 3.

To implement this algorithm and improve its computational efficiency, we use a data structure called *linked list* which is a type of list data structures. The linked list consists of a set of nodes representing a sequence. Every node is composed of a subsequence of data and link to the next node in the sequence. The linked list allows for efficient insertion or removal of nodes from any positions in the sequence. From this point of view, the linked list is effective in processing dynamic graphs, and allocating the needed memory. Moreover, the access time could be reduced when the input bipartite graph is cohesive.

The triangle set T of G is initialized as an empty set before starting the algorithm. The first loop processing from step 2 to step 12 is the same as the previous proposed quasi-k-truss decomposition algorithm. All nodes of G are visited once by adopting the standard breadth-first search algorithm. For the implementation, the breadth-first search is used for traversing the entire graph. The breadth-first search is a uninform search method aims at expanding and examing all nodes of a graph or combination of sequences by systematically searching through every solution. The search begins from the root node and explores all of the neighboring nodes in a graph. Then, for each of the nearest neighbor node, search their unexplored neighbor nodes which are reachable nodes in a graph. All nodes in a linked list are read in order from the starting point for linked list is inherently sequential access. The search stops until the last nearest neighbor node in the graph is visited. After traversing the entire graph, and matching all nodes, the auxiliary edges $e' \in E'$ is generated for constructing triangles in G. So the given bipartite graph G is transformed into G', where $G' = (V_1 \cup V_2, E \cup E')$. In the next loop processing, from step 13 to step 26, all of the edges $e \in E$ is traversed, but excluding the auxiliary edges $e' \in E'$ because the auxiliary edges $e' \in E'$ is only used for constructing triangles in the given bipartite graph G. The edge $e \in E$ are removed from the graph G recursively based on the number of triangles which containing the edge e. However, the number of triangles in which containing the edge e and its adjacency edges decreases. In here, a novel definition of edge $e \in E$ is proposed. It is the number of edges in which adjacent to the edge e, denoted by $C_{(e)}$ in the algorithm 3. For example, the triangle constructed by the nodes$\{v, v_j, v_k\}$ is deleted if the edge $e = (v, v_j)$ or $e = (v, v_k)$ is removed from G. Then, the value of C_e of edge $e = (v, v_j)$ or $e = (v, v_k)$ is $C_e - 1$. This process is proposed for preventing the drawback of edges' multiple traversal, and removing the sparse edges e from G'. The given value of parameter Q decreases when the value $C_{(e)}$ of any edge e is smaller than the value of parameter Q. The edge e contained in more number of triangles than the number of triangles containing its adjacency edges will be removed prioritily. Finally, the set of edges that contained in the cohesive subgraphs of G is extracted as the output result.

Algorithm 3: *Improved Quasi-k-truss Decomposition Algorithm*

- input graph G is transformed into G'
- T := a set of triangles
- q := queue for graph traversal
- e := an edge in $E = E(G)$
- e' := an auxiliary edge in $E' = E(G')$
- Q := input parameter for the number of triangles of an edge e
- $C_{(e)}$:= number of adjacency edges of an edge e

Input: $G = (V_1 \cup V_2, E)$, Q = 1,2,3,4....m
Output: T within Q hierarchy

```
1 begin
    T = φ
    for all v ∈ (V1 ∪ V2) do
        q.enqueue(v)
        while not q.empty() do
            v = q.dequeue()
            if (v, v_j), (v, v_k) ∈ E(G) then
                e' = (v_j, v_k) generated
                T = T ∪ {v, v_j, v_k}
            end
            transform G into G' = (V1 ∪ V2, E ∪ E')
        end
    end
    for all e ∈ G' do
        if e is contained in at least Q triangles then
            remove e from G'
        end
        for all edges adjacent to e do
            C_(e) = C_(e) − 1
        end
        if C_(e) < Q then
            Q = Q − 1
        end
        all edges in subgraphs of G
    end
    return
end
```

3.3 Complexity

The graph traverse starts from step 2 to 12 in algorithm 3. The computation time for this loop depends on the number of nodes in given G. A node is visited as the starting point. Then, the node is visited once again as the neighbor node of other nodes. Thus, every node in both V_1 and V_2 is visited twice while the edge e' is generated. Generally, the time cost is summarized by $O(2n)$ for the first loop processing. At the next loop from step 13 to 26, let $v_a \in V_1$, and $(v_j \cup v_k) \in V_2$. Then, $(e_{aj} \cup e_{ak}) \in T_{ajk}$ if $N_{(v_a)} \in (v_j \cap v_k)$. The necessary time cost for this loop from step 13 to 26 is $\sum_{e \in E} e$ to identify the number of triangles in a dense subgraph. Summarize the analysis, we could proof the total time complexity of algorithm 3 is $O(2n+m)$. Besides, the graph structure of a given bipartite graph G is extended after the generation of auxiliary edge $e' \in E'$. It needs $O(n^2)$ memory space to store the entire G' in main memory.

Chapter 4

Experiments and Evaluations

4.1 Triangle Listing in Bipartite Graphs

We observe the performance of the proposed algorithms through a succession of experiments. The experimental results verify the effectiveness of the quasi-k-truss decomposition algorithm in this section. All of the experiments are done on a machine with the Inter i7 2.3GHz CPU, 8GB RAM, and the version 4.1.2 C++ compiler in Mac OS 10.8.3.

Initially, we use eight datasets in bipartite structure in evaluating the effectiveness of triangle listing algorithm for bipartite graphs. These datasets recorded the relationship between the users and the hashtags of tweets in twitter social networks. In Table 4.1, the number of nodes and edges of each dataset are indicated by #$nodes$ and #$edges$ respectively. The number of triangles contained in the preprocessed bipartite graphs is indicated by #T. The memory usage and time cost for listing all triangles are indicated by *memory(in kilobyte)* and *time(in sec.)* respectively. The experimental results are concluded in Table 4.1. From these results, the time cost increases with the increasing number of triangles in the bipartite graphs. In addition, the number of triangles increases within a larger size of given bipartite graph. It proves the most difficult process in the proposed method is to match every node in two node sets, and generate auxiliary edges e' to build triangles in bipartite graphs. In this process, every node in both two node sets is visited at least twice when it is matching with other nodes or being matched with other nodes. One more important point should be stated that the number of enumerated triangles does not equal to the number of auxiliary edges e' because an auxiliary edge e' is usually contained in more than one triangle.

Table 4.1 Triangle listing for twitter network based on user-hashtag relation

dataset	$\#nodes$	$\#edges$	$\#T$	memory(in kilobyte)	time(in sec.)
input0311	195,129	386,579	1,186,184,782	403,092	4579.65
input0312	252,291	650,422	4,982,805,230	405,203	17120.64
input0313	197,694	324,701	180,172,382	402,597	1150.25
input0314	193,055	306,482	112,959,843	402,451	857.87
input0315	197,348	313,949	110,496,228	402,511	855.18
input0316	192,356	306,704	99,284,640	402,453	809.68
input0317	190,103	310,733	101,079,261	402,485	868.16
input0318	278,618	555,807	778,912,381	404,446	4044.97

According to the proposed method, the set of auxiliary edges e' is generated for triangle formation in a given bipartite graph. The relationship between the total number of triangles and the total number of auxiliary edges e' is observed. The eight twitter datasets are used in the related experiments. The Fig. 4.1 shows the relationship between the total number of triangles and the total number of auxiliary edges e'. In Fig. 4.1, the *X-axis* records the name of each dataset. The *Y-axis* records the metric of the number of triangles and the number of auxiliary edge e'. The graphs in blue indicate the total number of triangles in the given bipartite graphs. The graphs in red indicate the total number of auxiliary edges e' in the given bipartite graphs. The experimental results reveal that the total number of triangles does not directly related to the total number of auxiliary edges e' in a given bipartite graph. More triangles are formed with less auxiliary edges e', such as the results of the datasets *input0311* and *input0312*. Less triangles are formed with more auxiliary edges e', such as the results of the datasets *input0315* and *input0316*. These experimental results imply that the connectivity occurrence between two set of nodes in a bipartite graph plays a vital role in the number of triangles formed in a bipartite graph, but not the number of auxiliary edges e'.

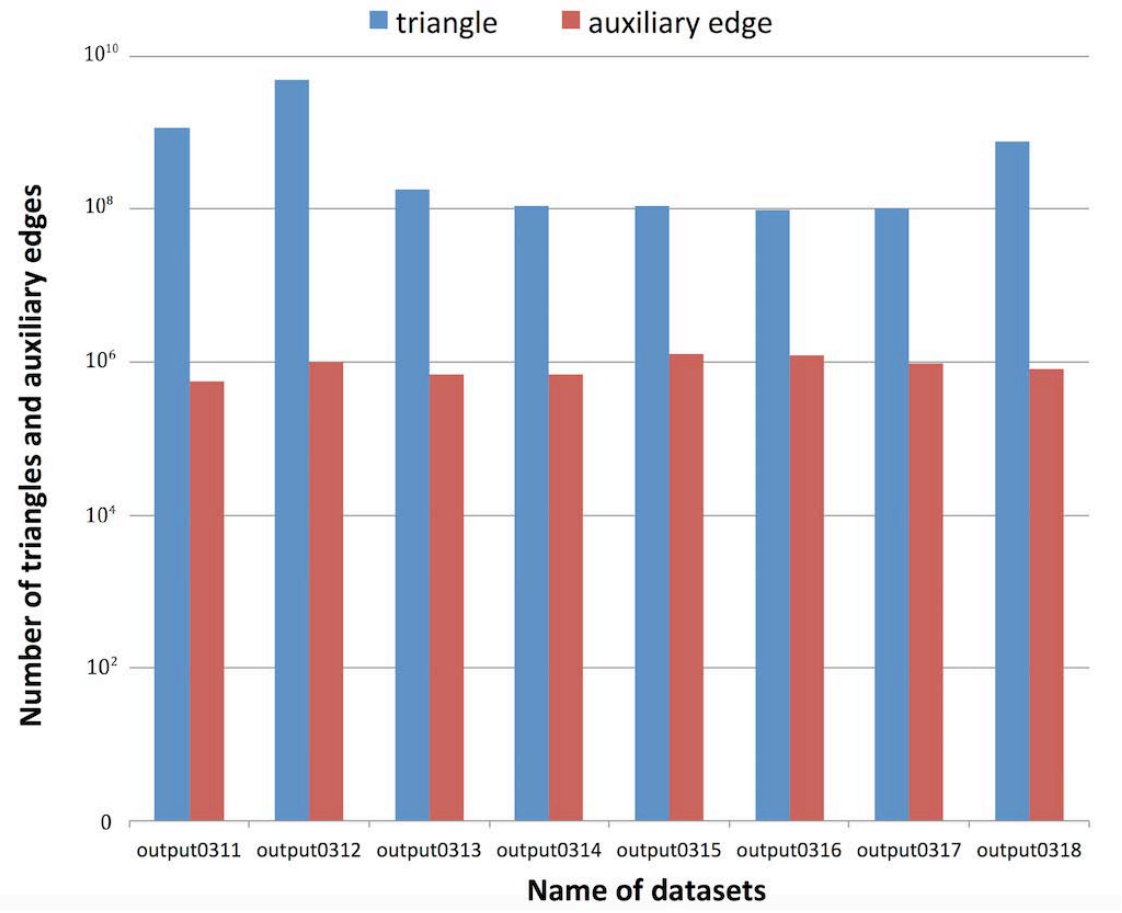

Fig. 4.1 The relationship between the number of triangles and the number of auxiliary edges

Next, we use other five real-world bipartite social networks within different sizes as the next experimental datasets. The Table 4.2 shows the features of these datasets. $|V_1|$ and $|V_2|$ represent the number of nodes of each node set in these given datasets. $|E|$ is the number of edges in each dataset. The three datasets *cmuDiff*, *cmuSame* and *cmuSim* are chosen from 20 newsgroups data packages referred in [12]. Each of them corresponds to a certain topic, and records the relationship between keywords and news documents. Both *HetRec* and *MovieLens* are datasets in which released from the framework of Information Heterogeneity and Fusion in Recommender Systems. The *HetRec* dataset records the relationship between online users and artists/musics from *Last.fm online music system* in the year 2011. The *MovieLens* dataset contains anonymous ratings by MovieLens users toward a certain number of movies in the year 2000.

Table 4.2 Features of bipartite social networks

dataset	$\|V_1\|$	$\|V_2\|$	$\|E\|$	size(in kilobyte)
cmuDiff	3,000	5,932	263,325	32.1
cmuSame	3,000	7,666	185,680	46.6
cmuSim	3,000	10,083	288,989	260.0
HetRec	9,372	6,257	26,232	259.0
MovieLens	3,706	6,040	1,000,209	40.1

Matrix blocking proposed in [12] is a community detection technique based on the connectivity occurrence among all nodes in G. Oppositely, the proposed algorithms in this research are designed for decomposing a given bipartite graph into dense subgraphs, and identifying the densest subgraph. Therefore, in these experiments, we mainly observed three aspects to evaluate the proposed algorithm. First, we list all triangles in each bipartite social network as the previous experiments in Table 4.1. Second, we observe the time cost for enumerating all triangles in each bipartite graph within different given value of parameter.

The Table 4.3 records the experimental results by using the five social network datasets. The $\#T$ indicates the total number of triangles formed in each given bipartite social network. The time cost for forming all triangles is recorded within the unit of second.

Similar to the previous triangle listing experimental results, the time cost increases with the increasing number of triangles. Meanwhile, the number of auxiliary edges $e' \in E'$ also increases. But it is worth to notice that the number of auxiliary edges $e' \in E'$ does not equal to the total number of triangles for a auxiliary edge e' is usually contained in more than one triangle. These results show the connectivity occurrence in a bipartite graph has significant impact on the density of subgraphs.

Table 4.3 Triangle listing in bipartite social networks

dataset	#*T*	size(in kilobyte)	time(in sec.)
cmuDiff	61,638	874	0.374
cmuSame	174,363	2,458	0.780
cmuSim	1,838,827	27,471	7.207
HetRec	945,043	15,020	6.739
MovieLens	63,271	891	0.453

4.2 Triangle Count for Bipartite Graphs Decomposition

Another reasonable evaluation strategy is to observe the number of triangles within different given value of parameter. These given bipartite graphs are decomposed based on counting the number of triangles within different value of parameter. The time cost and memory usage consumption are observed as evaluational metrics. The Table 4.4 records the number of triangles of each given dataset within different value of parameter Q. The value of parameter Q is chosen as $\{1, 10, 20, 30, 40, 50\}$. From the results shown in the Table 4.4, the number of triangles decreases with the increasing value of parameter.

Table 4.4 The number of triangles within different value of parameter

dataset	Q					
	1	10	20	30	40	50
cmuDiff	61,638	12,162	3,700	2,846	2,459	2,331
cmuSame	174,363	23,440	9,325	5,334	4,326	2,830
cmuSim	1,838,827	6,354	2,481	1,589	1,457	1,095
HetRec	945,043	4,800	1,534	611	215	122
MovieLens	63,271	14,044	5,462	3,557	2,733	1,975

The time cost for counting the number of triangles within different value of parameter is shown in the Fig. 4.2. The *X-axis* of Fig. 4.2 records the value of given parameters. The *Y-axis* records the computational time cost. The time cost is recorded in the unit of second. Differ from the relationship between the number of triangles and the given parameter, the time cost ranges in a very narrow area even though the number of triangles decreases. This result implies that the time cost does not related to the number of triangles and the value of parameter, but depends on the sizes of bipartite graphs and the iterative process.

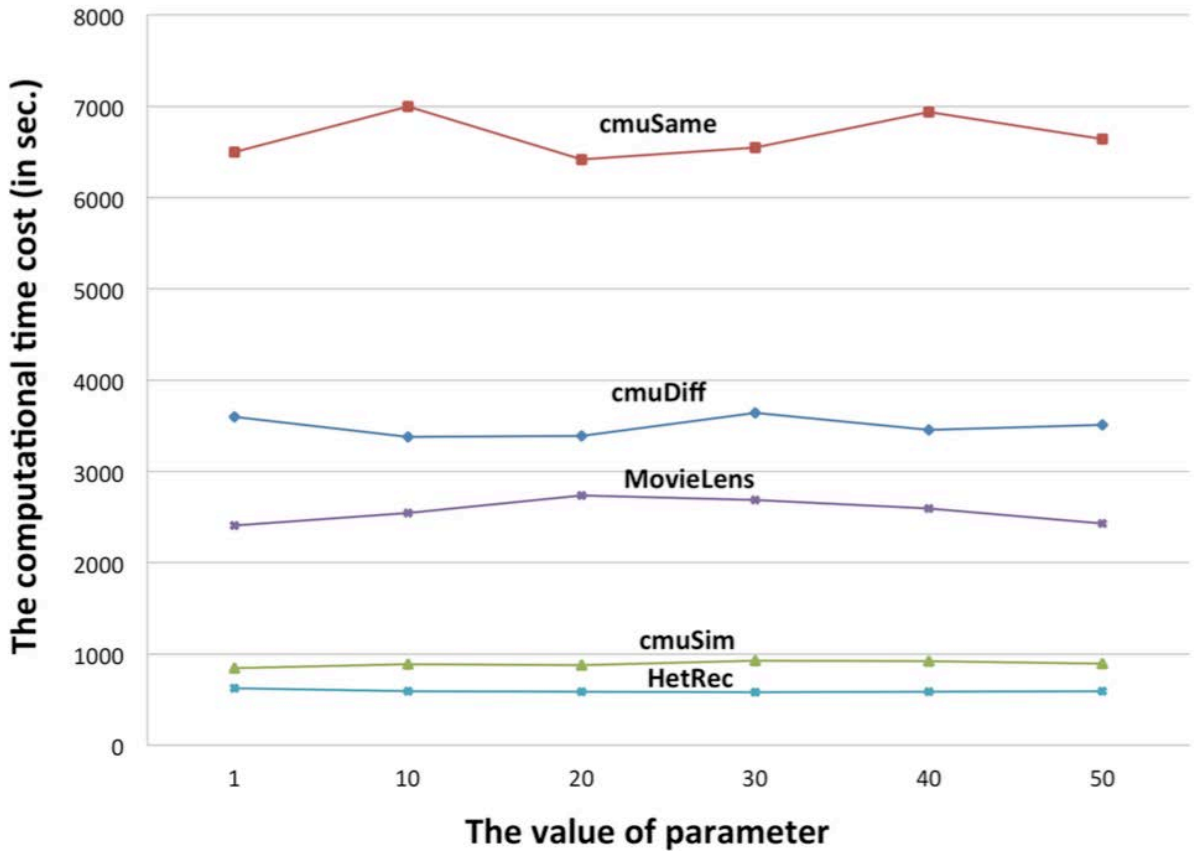

Fig. 4.2 The time cost for counting the number of triangles

Besides, the Fig. 4.3 shown the observed results of memory usage for counting the triangles in each dataset. The *X-axis* of Fig. 4.3 records the different given value of parameter. The *Y-axis* records the memory usage in the unit of byte. The results of memory usage is similar to the results of time cost. The memory usage does not related to the number of triangles, but depends on the sizes of bipartite graphs and the iterative process.

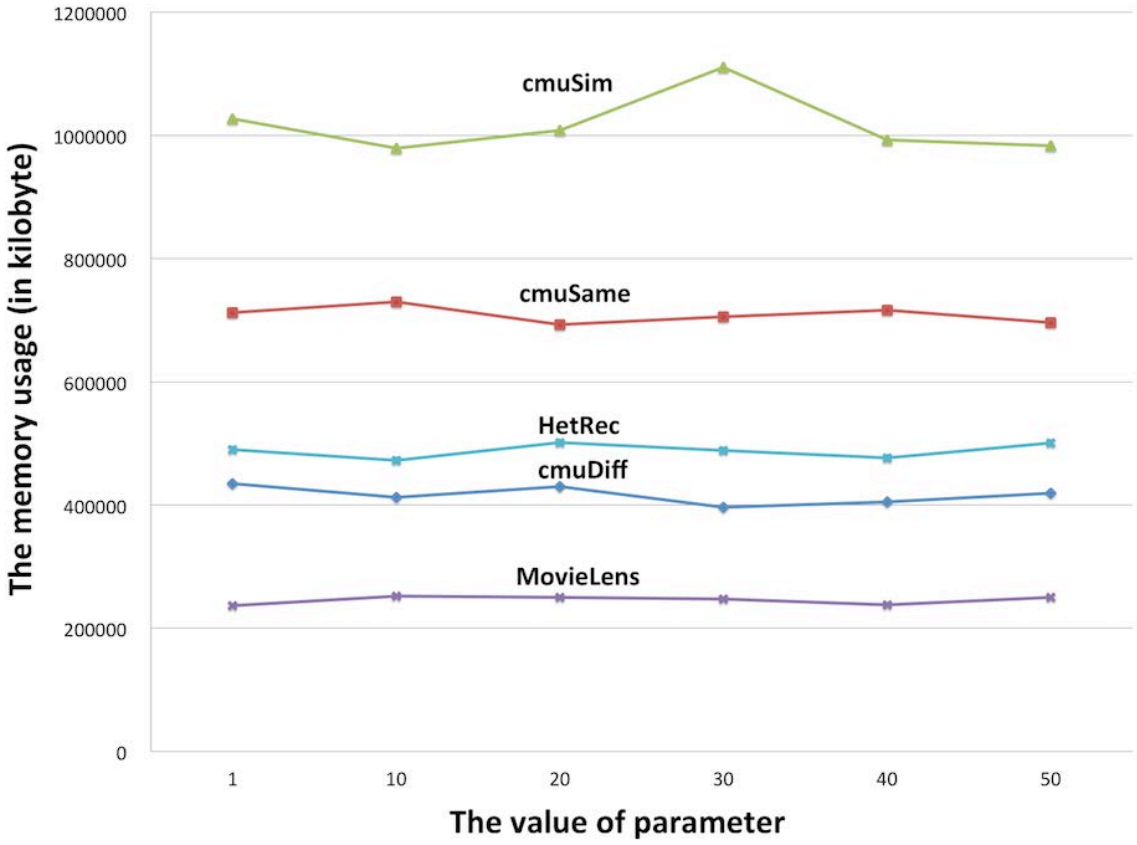

Fig. 4.3 The memory usage for counting the number of triangles

4.3 Observation on Density

The importance of connectivity occurrence between two set of nodes in a bipartite graph has been revealed by experimental results of the Fig. 4.1. Therefore, we employ the density for bipartite graphs to evaluate the quanlity of extracted dense subgraphs. We pick up the *cumDiff* dataset from the 20 newsgroup datasets to observe the relationship between the size of subgraphs and the number of subgraphs. The Fig. 4.4 shows the number of subgraphs extracted by the previous quasi-k-truss decomposition algorithm. In the Fig. 4.4, the *X-axis*

denotes the size of subgraphs, and the *Y-axis* denotes the number of extracted subgraphs. The result shows that the old version of quasi-k-truss decomposition algorithm can extract few subgraphs within small sizes. Then, we use the improved quasi-k-truss decomposition algorithm to extract subgraphs from the same dataset. The results are shown in the Fig. 4.5. The result shows that the improved quasi-k-truss decomposition algorithm can significantly extract more subgraphs than the old version algorithm.

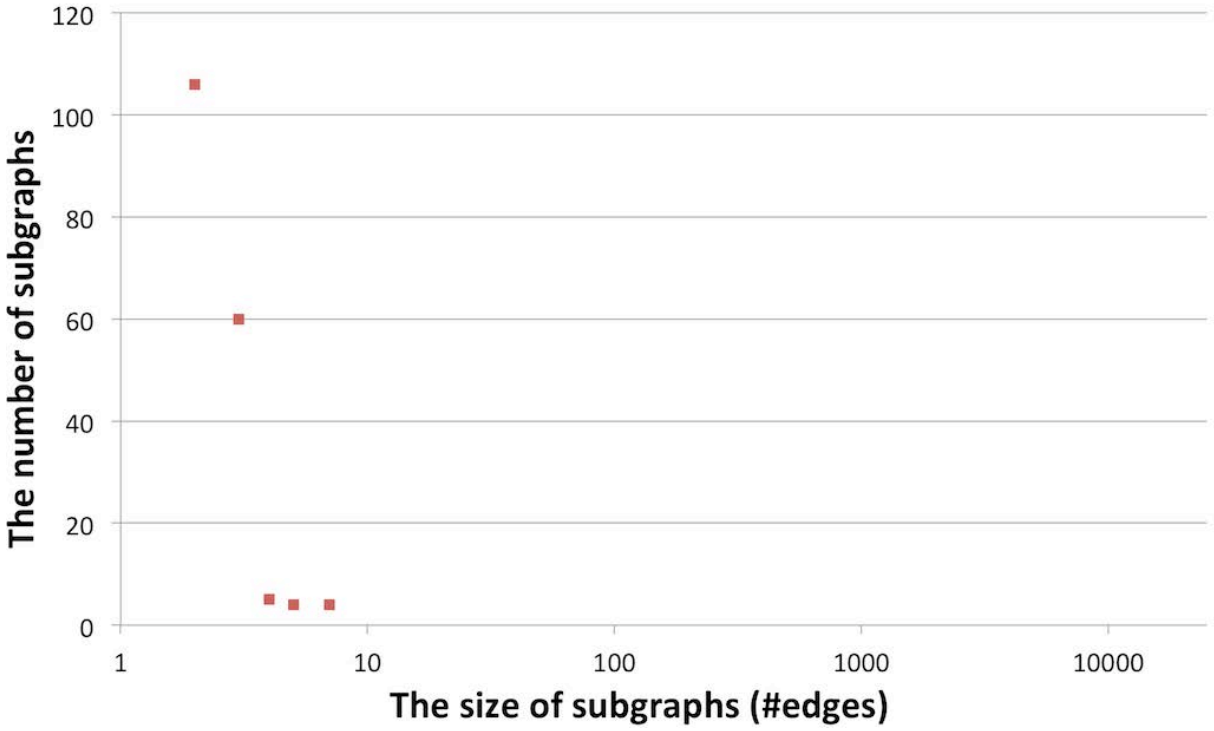

Fig. 4.4 The subgraph extraction result by the previous algorithm

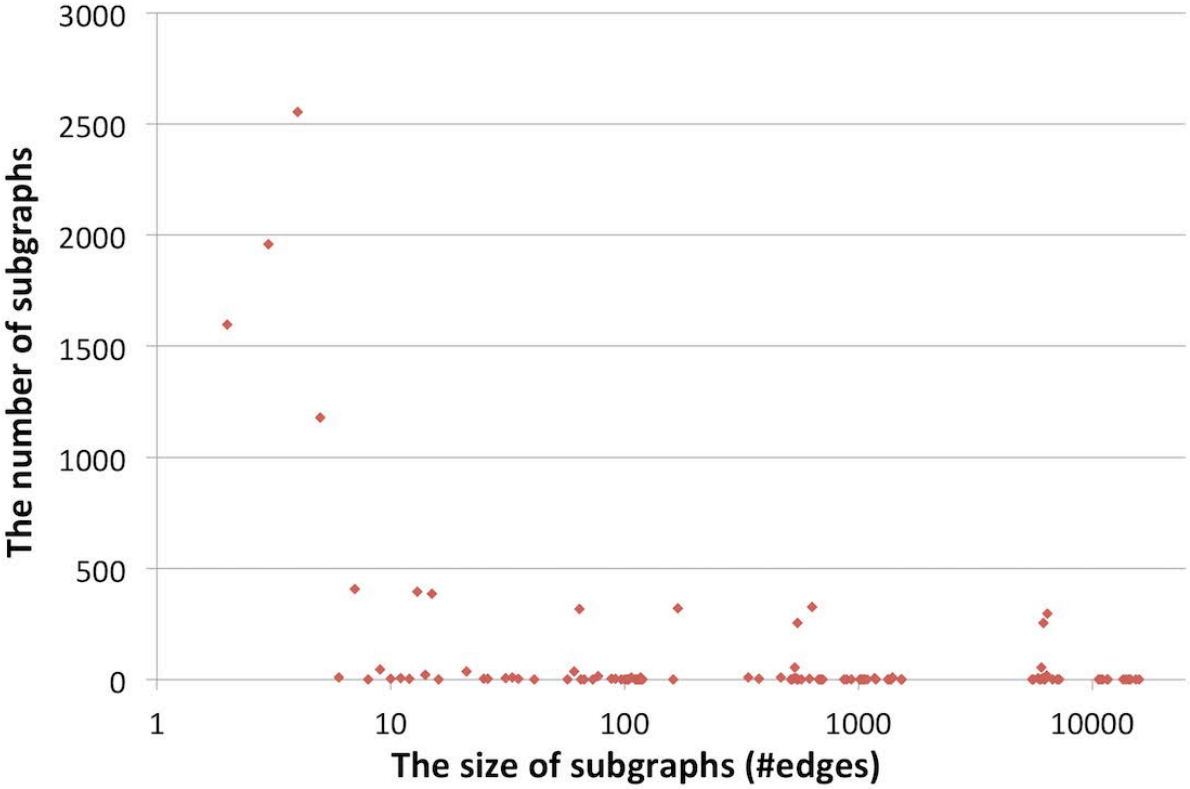

Fig. 4.5 The subgraph extraction result by the improved algorithm

The Fig. 4.6 shows the hierarchical distribution of dense subgraphs of *cumDiff* dataset with all possible value of k. These results are obtained by using the improved quasi-k-truss decomposition algorithm. The *X-axis* of the Fig. 4.6 records the sizes of dense subgraphs. The *Y-axis* records the value of k of dense subgraphs. The dense subgraphs within sizes smaller than 100 distribute in all levels in the given bipartite graph. The value of k of dense subgraphs with sizes smaller than 100 ranges from 1 to 341, where $1 < k < 341$. The dense subgraphs within larger value of k containing less number of edges in more triangles. These kinds of dense subgraphs distribute in higher hierarchy of the given bipartite graph with relatively high density. On the other hand, the dense subgraphs with sizes larger than 100 mainly located in lower level, where $1 < k < 178$. Thus, the set of edges belongs to those dense subgraphs within sizes larger than 100 is contained in less number of triangles. The density of those dense subgraphs within sizes larger than 100 is relatively low.

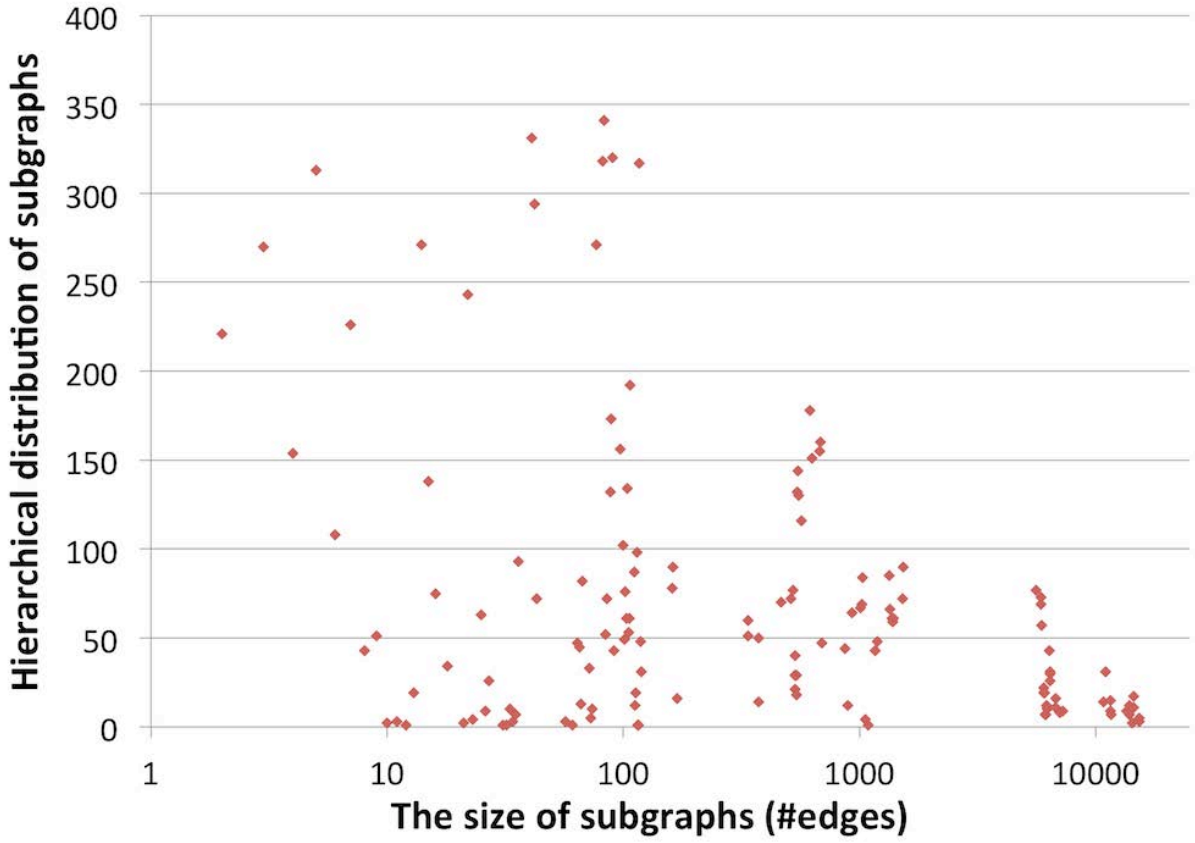

Fig. 4.6 Hierarchical distribution of dense subgraphs

Finally, we employ the density to evaluate the quality of extracted subgraphs. The formula for calculating the density of bipartite graphs is shown as below.

$$d_G = \frac{|E|}{|V_1||V_2|}$$

In the Fig. 4.7, the *X-axis* is the size of subgraphs. The *Y-axis* is the density of subgraphs. The red graph in the Fig. 4.7 indicates the extracted dense subgraphs by the improved quasi-k-truss decomposition algorithm. The result illustrates the efficiency of the improved algorithm in extracting dense substructures from bipartite graphs. Some of the extracted subgraphs have higher density with smaller sizes. Some of the extracted subgraphs have lower density with larger sizes. On the contrary, in order to verify the efficiency of the improved algorithm in extracting dense subgraphs as comparison, we compute the density of sparse substructure by using the improved algorithm. The blue graph in the Fig. 4.7 indicates the density of sparse substructures of the bipartite graph. The density of sparse subgraphs decreases due to the weak connection between two sets of nodes remained in the sparse substructure. The larger size of the sampled sparse substructure, the lower density because of the weak connection. The smaller size of the sampled substructure, the higher density because of the strong connection in the substructure. Compare to the density of extracted dense subgraphs, the density of the sparse substructures is low. These results prove the efficiency of the improved algorithm in extracting dense subgraphs from bipartite graphs.

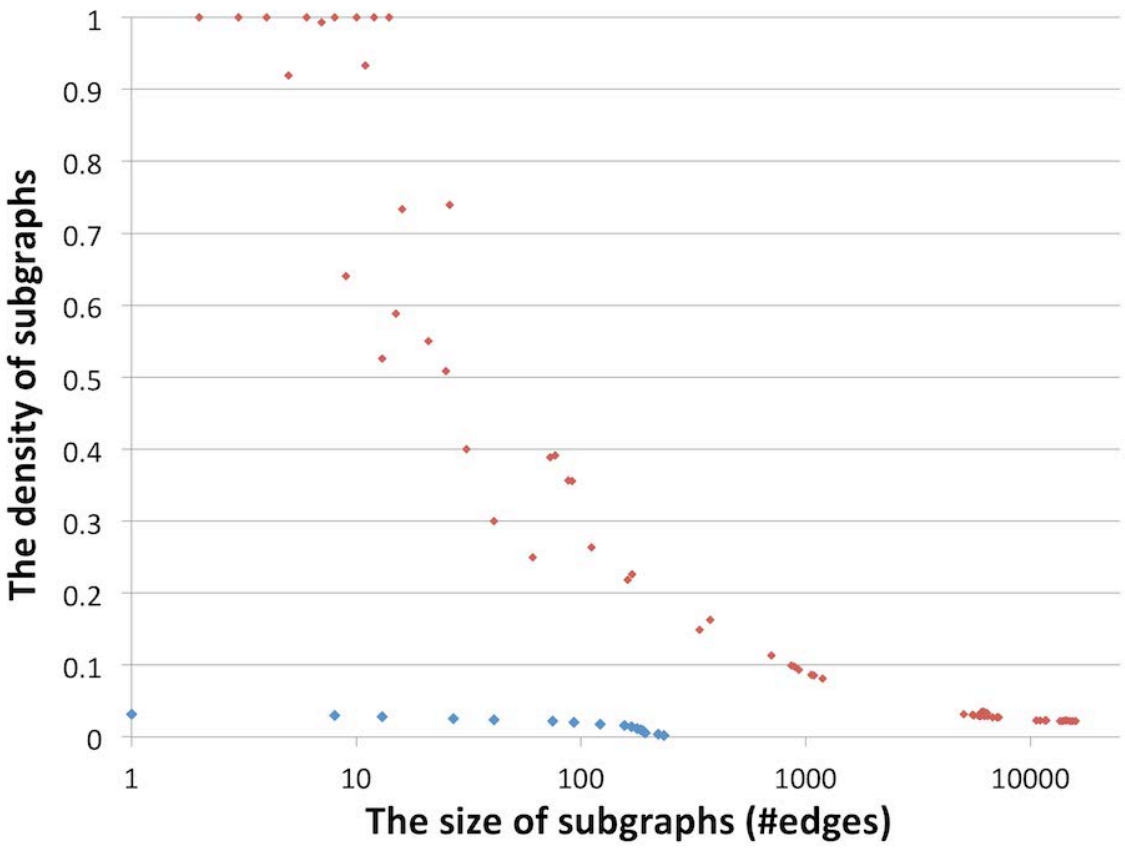

Fig. 4.7 Comparison of the density of subgraphs by the improved algorithm and random sampling

4.4 Implementation of k-core Decomposition Algorithm

As one of well-known graph decomposition algorithm, the k-core decomposition algorithm has been applied in various industrial fields, such as graph decomposition for graph data mining [2], grahp visualization [26], and fingerprinting system [27]. The k-core decomposition algorithm is a node degree distribution based decomposition algorithm. A given graph G can be decomposed based on the degree of nodes so that the dense subgraphs in every hierarchy are found where the degree of nodes in dense subgraphs is larger than k, denoted by $deg_{(v)} > k$. The extracted dense subgraphs represent the "*core*" of a graph G.

We implement the k-core decomposition algoriothm with purpose of comparing the efficiency of dense subgraph extraction between the proposed algorithm in this work and the k-core decomposition algorithm. The data structure of linked list is employed to implement the k-core decomposition algorithm. A node of G is focused, and its adjacent nodes are connected with it by pointer. A focused node of G in the linked list consists of a data value and a pointer pointing at its adjacent nodes. The focused node and its adjacent nodes that together represent a sequence of the linked list. The degree of each node can be computed by counting the number of its adjacent nodes. At the same time, the connectivity occurrence between two set of nodes V_1 and V_2 can also be detected by using the linked list data structure. Every node is focused once for searching out all nodes directly connect to it. The degree of its adjacent nodes is minus one if a focused node is completely removed out of the linked list. A group of nodes that are mutural neighbor nodes in the linked list represents a dense subgraph of G. All nodes of the dense subgraph have a certain degree larger than k. A given bipartite graph G can be decomposed based on the node degree. The denset subgraph represents the "*core*" of G can be extracted.

We mainly observe three aspects as the experimental results: the degree of the subgraphs, the size of the subgraphs and the number of subgraphs. The dataset of *cmuDiff* is also used in these experiments. The Fig. 4.8 shows the number of dense subgraphs extracted by using the implemented k-core decomposition algorithm. The *X-axis* records the size of the dense subgraphs. The *Y-axis* records the number of dense subgraphs. The total number of dense subgraphs extracted by the k-core decomposition algorithm is less than the total number of dense subgraphs extracted by the previous version of quasi-k-truss decomposition algorithm. Compare to the dense subgraph extraction result shown in the Fig. 4.4, the k-core decomposition algorithm extracts more large size dense subgraphs than the previous version of quasi-k-truss decomposition algorithm. However, the improved quasi-k-truss decomposition algorithm extracts more dense subgraphs within large sizes than the number of dense subgraphs extracted by the k-core decomposition algorithm. Through comparing the number of dense subgraphs extracted by the k-core decomposition algorithm

and the improved quasi-k-truss decomposition algorithm, extracting dense subgraphs based on triangle structure is more effective than the degree of nodes due to the full connection among three nodes in a triangle. The dense subgraph extraction result proves the efficiency of the improved quasi-k-truss decomposition algorithm in dense subgraph extraction.

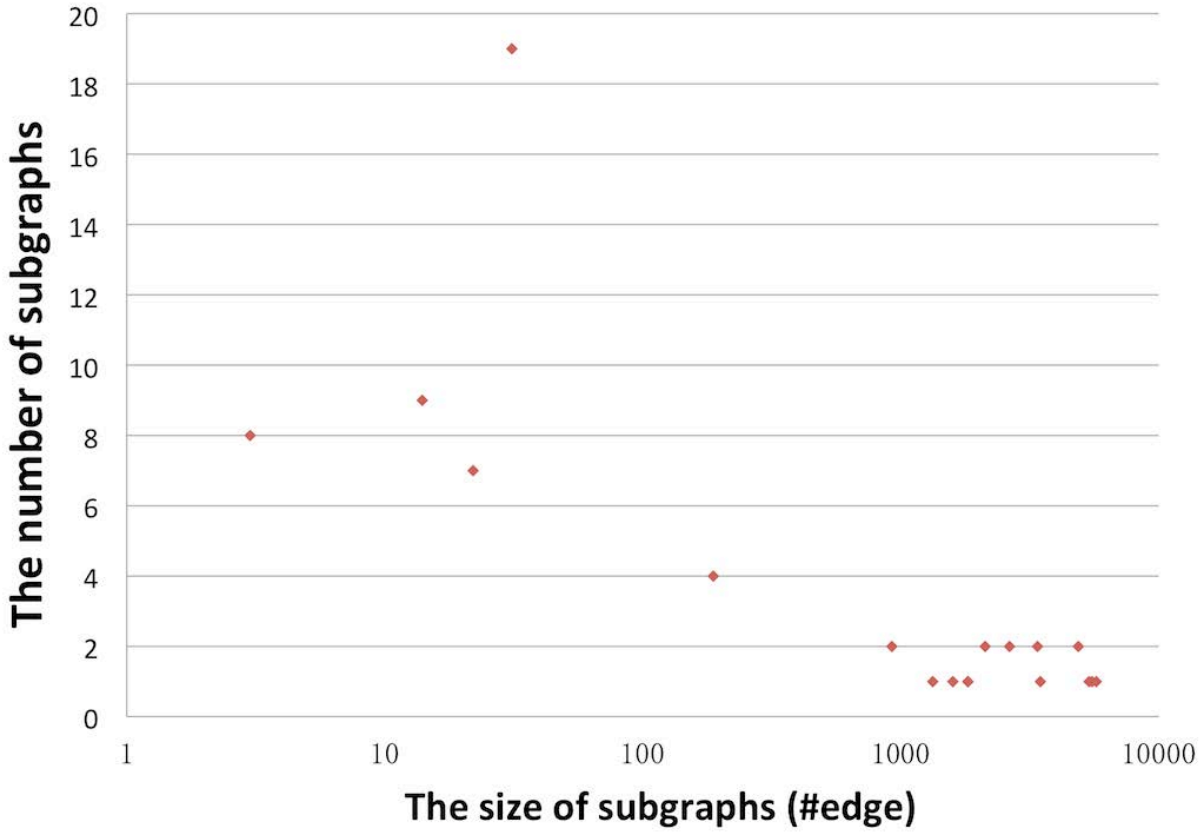

Fig. 4.8 The subgraph extraction by the k-core decomposition algorithm

All nodes of a dense subgraph are extracted based on a certain degree, where $deg_{(v)} > k$. We observe the distribution of k value for the extracted dense subgraphs in order to find out the location of extracted dense subgraphs in the graph G. The Fig. 4.9 illustrates the distribution of k value of the extracted dense subgraphs. The *X-axis* records the size of the dense subgraphs. The *Y-axis* records the distribution of k value of dense subgraphs. Consider the size 100 as the dividing line, all of the dense subgraphs within sizes larger than 100 has small value of k. On the other hand, those dense subgraphs within sizes smaller than 100 have large value of k. This result proves that most of large size dense subgraphs are located in low hierarchy of the graph G due to the low degree of nodes of the large size dense subgraphs. The small size dense subgraphs are located in high hierarchy of the graph G. This result matches the hierarchical distribution of dense subgraphs resulted by the improved quasi-k-truss decomposition algorithm shown by the Fig. 4.6.

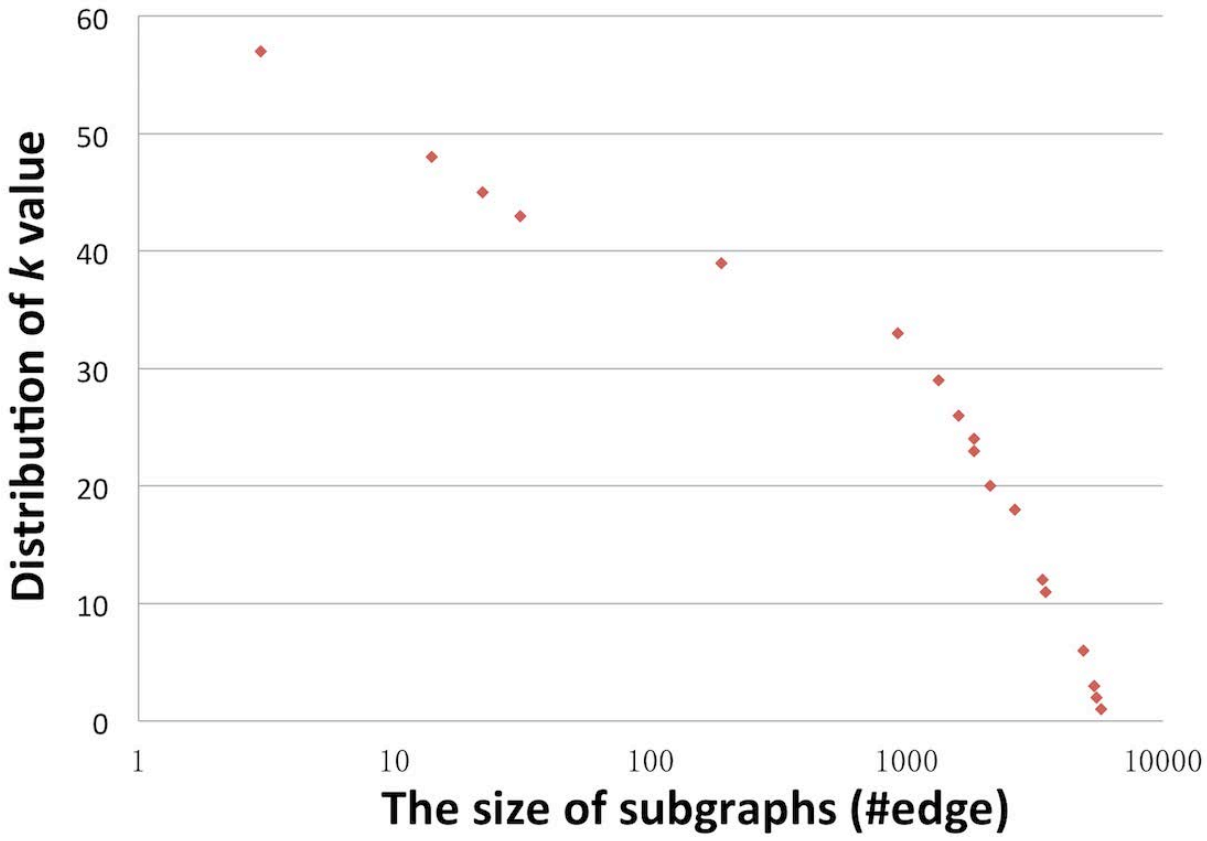

Fig. 4.9 Distribution of k value of dense subgraphs

Chapter 5

Conclusion and Future Work

The research concluded in this dissertation mainly focuses on algorithmic design for dense subgraphs extraction in bipartite graphs. The k-truss decomposition algorithm is referred. The notion of k-truss is extended to bipartite graphs. The extended k-truss is named quasi-k-truss, and defined as the densest subgraphs of bipartite graphs. It represents community in bipartite graphs. Similar to the notion of k-truss, every edge of quasi-k-truss is contained in at least $(k-2)$-triangles, such that dense subgraphs can be extracted from bipartite graphs based on triangle.

Due to the special graph structure of bipartite graphs, the community extraction algorithms for bipartite graphs are limited. On the other hand, the k-truss decomposition algorithm is not applicable to bipartite graphs for no triangles are contained in bipartite graphs. Thus, we propose the auxiliary edge to build triangles in bipartite graphs. The truss-like components represent communities in bipartite graphs can be extracted based on triangle. Three algorithms are designed to approach the achievement step by step. Numerous data structures, such as array, hash table and list, are employed to implement the proposed algorithms in order to improve the computational efficiency.

The succession of experimental results prove the effectiveness of the quasi-k-truss decomposition algorithm in extracting communities from bipartite graphs based on triangle. Especially the improved algorithm is significantly efficiency in extracting communities within high density from bipartite graphs. However, some problems should be pointed out for a brief discussion. First, with the increasing number of auxiliary edges in a given bipartite graph, the iterative process increases. A novel technique for pruning the auxiliary edges is important to reduce the iterative process. Second, with more triangles anchor in a given bipartite graph, the sizes of bipartite graphs become exceedingly large. To hold the entire graphs with all triangles, large memory space of computer mainframe is necessary. Third, it is intractable to reduce the time cost. The experimental results reveal that the time cost

increases with the sizes and the cohesiveness of the input bipartite graphs. To reduce the time cost, available data structure is crucial to simplify the iterative procedures. Finally, evaluation based on large empirical graphs is proved challenging for it is difficult to define a criterion for the desired communities. This challenge leads to a lack of evaluation methods for massive graphs. In this research, we employ the graph density as criterion to evaluate the quality of extracted communities and the efficiency of the proposed algorithm. However, it is worthy of refining the measurements and evaluation of the extracted communities.

The theories and algorithms we concluded in this dissertation is still preliminary and imperfect. It it necessary to be extended and intensified. As the future work, the mathematical modeling for the quasi-k-truss and the time complexity analysis of the proposed algorithms are crucial to be improved as the theoretical study. Moreover, as aforementioned, simplify the data structure with purpose of reducing the computation time cost is necessary.

References

[1] Abello J., Resende M.G.C., and Sudarsky S. Massive quasi-clique detection. LATIN2002, pp.598-612, 2002.

[2] Batagelj V., and Zaversnik M. An $O(m)$ algorithm for cores decomposition of networks. arXiv preprint cs/0310049, Vol. 5, No. 2, pp.129-145, 2003.

[3] Cohen J. Truss: Cohesive subgraphs for social network analysis. National security agency technical report, pp.1-29, 2008.

[4] Cohen J. Graph twiddling in a map reduce world. Computing in science and engineering, Vol. 11, No. 4, pp.29-41, 2009.

[5] Chu S., and Cheng J. Triangle listing in massive networks and its applications. The proceedings of the 17^{th} international conference on Knowledge discovery and data mining, pp.672-680, ACM SIGKDD 2011.

[6] Cook D.J., and Holder L.B. Mining graph data. John Wiley & Sons, pp.443-468, 2006.

[7] Cheng J., Ke Y., Chu S., and Ozsu M.T. Efficient core decomposition in massive networks. The proceedings of 27^{th} international conference of data engineering on IEEE, pp.51-62, ICDE 2011.

[8] Chua F.C.T., and Lim E.P. Modeling bipartite graphs using hierarchical structures. Advances in social networks analysis and mining international conference on IEEE, pp.94-101, ASONAM 2011.

[9] Cheng K., Li Y.T., and Wang X. Single Document Summarization Based on Triangle Analysis of Dependency Graphs. The proceedings of 16^{th} international conference of network-based information systems, pp.38-43, NBiS 2013.

[10] Csermely P., London A., Wu L.Y., and Uzzi B. Structure and dynamics of core/periphery networks. Journal of complex networks, 1.2, pp.93-123, 2013.

[11] Clauset A., Newman M.E., and Moore C. Finding community structure in very large networks. Physical review E, Vol.70, pp.066111-066117, 2004.

[12] Chen J., and Saad Y. Dense subgraph extraction with application to community detection. Knowledge and data engineering on IEEE Transaction, Vol. 24, Issue: 7, pp.1216-1230, 2012.

[13] Charalampos E., and Tsourakakis. Fast counting of triangles in large real networks without counting: algorithms and laws. The 8^{th} international conference of ICDM on IEEE, pp.608-617, 2008.

[14] Danon L., Diaz-Guilera A., Duch J., and Arenas A. Comparing community structure identification. Journal of statistical mechanics: theory and experiment, Vol.2005, No.09, pp.09008-09017, 2005.

[15] Dorogovtsev S.N., Goltsev A.V., and Mendes J.F.F. K-core organization of complex networks. Physical review letters, 96.4, pp.229-252, 2006.

[16] Dourisboure Y., Geraci F., and Pellegrini M. Extraction and classification of dense communities in the web. pp.461-470, IW3C2 2007.

[17] Everett M.G., and Borgatti S.P. Analyzing clique overlap. Connections, Vol. 21, No.1, pp.49-61, 1998.

[18] Freeman L.C. The development of social network analysis: a study in the sociology of science. Empirical press, Vol. 1, pp.12-59, 2004.

[19] Fortunato S. Community detection in graphs. arXiv:0906.0612v2, physics.soc-ph, pp.75-174, 2010.

[20] Fiedler M., and Borgelt C. Support computation for mining frequent subgraphs in a single graph. The 5^{th} international conference on mining and learning with graph, 2007.

[21] Feng J., He X., Konte B., Bohm C., and Plant C. Summarization-based mining bipartite graphs. The proceedings of the 18^{th} international conference on knowledge discovery and data mining, pp.1249-1257, ACM SIGKDD 2012.

[22] Flake G.W., Lawrence S., and Giles C.L. Efficient identification of web communities. The proceedings of Knowledge discovery and data mining, pp.150-160, 2000.

[23] Flake G.W., Lawrence S., Giles C.L., and Coetzee F.M. Self-organization and identification of web communities. Computer, 35.3, pp.66-70, 2002.

[24] Gibson D., Kleinberg J., and Raghavan P. Inferring web communities from link topology. The proceedings of the 9^{th} international conference on hypertext and hypermedia, pp.225-234, ACM 1998.

[25] Girvan M., and Newman M.E.J. Community structure in social and biological networks. The proceedings of the national academy of sciences, pp.7821-7826, 2002.

[26] Hamelin A., Ignacio J., Dall'Asta L., Barrat A., Vespignani A. k-core decomposition: A tool for the visualization of large scale networks. arXiv preprint cs/0504107, 2005.

[27] Hamelin A., Ignacio J., Dall'Asta L., Barrat A., Vespignani A. Large scale networks fingerprinting and visualization using the k-core decomposition. In Advances in neural information processing systems, pp. 41-50, 2005.

[28] Kleinberg J.M. Authoritative sources in a hyperlinked environment. SODA 1998, pp.668-677, 1998.

[29] Kleinberg J.M. Authoritative sources in a hyperlinked environment. Journal of the ACM, 46.5, pp.604-632, 1999.

[30] Kuramochi M., and Karypis G. Frequent subgraph discovery. The proceedings of ICDM International Conference on IEEE, pp.313-320, 2001.

[31] Kumar R., Raghavan P., Rajagopalan S., and Tomkins A. Extracting large-scale knowledge bases from the web. VLDB1999, pp.639-650, 1999.

[32] Knoke D., and Yang S. Social network analysis. Sage publications, Vol. 154, pp.5-11, 2008.

[33] Latapy M. Main-memory triangle computations for very large (sparse (power-law)) graphs. Theoretical computer science, 407.1, pp.458-473, 2008.

[34] Li Y.T., and Cheng K. Keyflow: viewpoint extraction based on word frequency and dependency relations. Forum on information technology 2010, pp.181-182, 2010.

[35] Li Y.T., and Cheng K. Single document summarization based on clustering coefficient and transitivity analysis. Data engineering and information management 2011.

[36] Lee C., and Cunningham P. Community detection: effective evaluation on large social networks. Journal of complex networks, comnet/cnt012, pp.19-37, 2013.

[37] Liu X., and Murata T. Community detection in large-scale bipartite networks. The proceedings of web intelligence and intelligent agent technologies on IEEE 2009, pp.50-57, WI-IAT 2009.

[38] Li Y.T., Maeda K., Kuboyama T., and Sakamoto H. An extension of community extraction algorithm on bipartite graph. The international conference on advances in computer and electronics technology 2014, 2014.

[39] Li Y.T., Kuboyama T., and Sakamoto H. Mining twitter data: discover quasi-truss from bipartite graph. The 5^{th} international conference on intelligent decision technologies 2013, pp.287-293, KES-IDT 2013.

[40] Li Y.T., Kuboyama T., and Sakamoto H. An implementation of truss decomposition of bipartite graph. DDS 2013, 2013.

[41] Li Y.T., Kuboyama T., and Sakamoto H. Truss decomposition for extracting communities in bipartite graph. The 3^{rd} international conference on advances in information mining and management 2013, pp.76-80, IMMM 2013.

[42] Li Y.T., and Sakamoto H. The implementation of triangle counting based on random sampling. pp.13-15, SIG-FPAI 2013.

[43] Mao C. A heuristic algorithm for bipartite community detection in social networks. Journal of software, 7.1, pp.204-211, 2012.

[44] Matsuda H., Ishihara T., and Hashimoto A. Classifying molecular sequences using linkage graph with their pairwise similarities. Theor. Compt. Sci., Vol. 210, No. 2, pp.305-325, 1999.

[45] Montresor A., Pellegrini F.D., and Miorandi D. Distributed k-core decomposition. Parallel and distributed systems on IEEE Transactions, 24.2, pp.288-300, 2013.

[46] Newman M.E.J. Networks: an introduction. Oxford university press, pp.371-372, 2010.

[47] Newman M.E.J. Detecting community structure in networks. The european physical journal b-condensed matter and complex systems, Vol. 38, No. 2, pp.321-330, 2004.

[48] Nooy D.W., Mrvar A., and Batagelj V. Exploratory social network analysis with pajek. Cambridge university press, Vol. 27, pp.3-29, 2011.

[49] Page L., Brin S., Motwani R., and Winograd T. The PageRank citation ranking: bringing order to the web. Stanford digital libraries working paper, Vol. 72, pp.1-28, 1999.

[50] Palla G., Derenyi I., Farkas I., and Vicsek T. Uncovering the overlapping community structure of complex networks in nature and society. Nature, 435.7043, pp.814-818, 2005.

[51] Pei J., Jiang D., and Zhang A. On mining cross-graph quasi-cliques. pp.228-238, SIGKDD 2005.

[52] Porter M.A., Onnela J.P., and Mucha P.J. Communities in networks. Notices of the AMS, 56.9, pp.1082-1097, 2009.

[53] Pagh R., and Tsourakakis C.E. Colorful triangle counting and a mapreduce implementation. The information processing letters, 112.7, pp.277-281, 2012.

[54] Radicchi F., Castellano C., Cecconi F., Loreto V., and Parisi D. Defining and identifying communities in networks. The proceedings of the national academy of sciences of the united states of america, Vol. 101, No. 9, pp.2658-2663, 2004.

[55] Scott J. Social network analysis: a handbook, 2nd edition. Sage publications, London, pp.7-46, 2000.

[56] Seidman S.B. Network structure and minimum degree. Social networks, Vol. 5, No. 3, pp.269-287, 1983.

[57] Schank T. Algorithmic aspects of triangle-based network analysis. Phd in computer science, University of Karlsruhe, pp.5-121, 2007.

[58] Satuluri V., and Parthasarathy S. Scalable graph clustering using stochastic flows: applications to community discovery. The proceedings of the 15^{th} international conference on Knowledge discovery and data mining, pp.737-746, ACM SIGKDD 2009.

[59] Suzuki K., and Wakita K. Extracting multi-facet community structure from bipartite networks. The proceedings of computational science and engineering international conference on IEEE, CSE'09, Vol. 4, pp.312-319, 2009.

[60] Tsourakakis C.E., Kang U., Miller G.L., and Faloutsos C. Doulion: counting triangles in massive graphs with a coin. The proceedings of the 15^{th} international conference on Knowledge discovery and data mining, pp.837-846, ACM SIGKDD 2009.

[61] Tao Y., Sheng C., and Li J. Finding maximum degrees in hidden bipartite graphs. The proceedings of the international conference on management of data, pp.891-902, ACM SIGMOD 2010.

[62] West D.B. Introduction to graph theory. Upper saddle river: prentice hall, Vol. 2, pp.11-70, 2001.

[63] Wasserman S. Social network analysis: methods and applications. Cambridge university press, Vol. 8, pp.3-49 1994.

[64] Wang J., and Cheng J. Truss decomposition in massive networks. The proceedings of the VLDB endowment, 5.9, pp.812-823, 2012.

[65] Watts D.J., Dodds P.S., and Newman M.E.J. Identity and search in social networks. Science, 296, pp.1302-1305, 2002.

[66] Watts D.J., and Strogatz S.H. Collective dynamics of "small-world" networks. Nature, 39, pp.440-442, 1998.

[67] Zhao Y.P., Levina E., and Zhu J. Community extraction for social networks. The proceedings of the national academy of sciences, 108.18, pp.7321-7326, 2011.

[68] Zha H., He X., Ding C., Simon H., and Gu M. Bipartite graph partitioning and data clustering. The proceedings of the 10^{th} international conference on information and knowledge management, pp.25-32, ACM 2001.

[69] Zhang P., Wang J., Li X., Li M., Di Z., and Fan Y. The clustering coefficient and community structure of bipartite networks. Physical a: statistical mechanics and its applications, 387.27, pp.6869-6875, 2008.